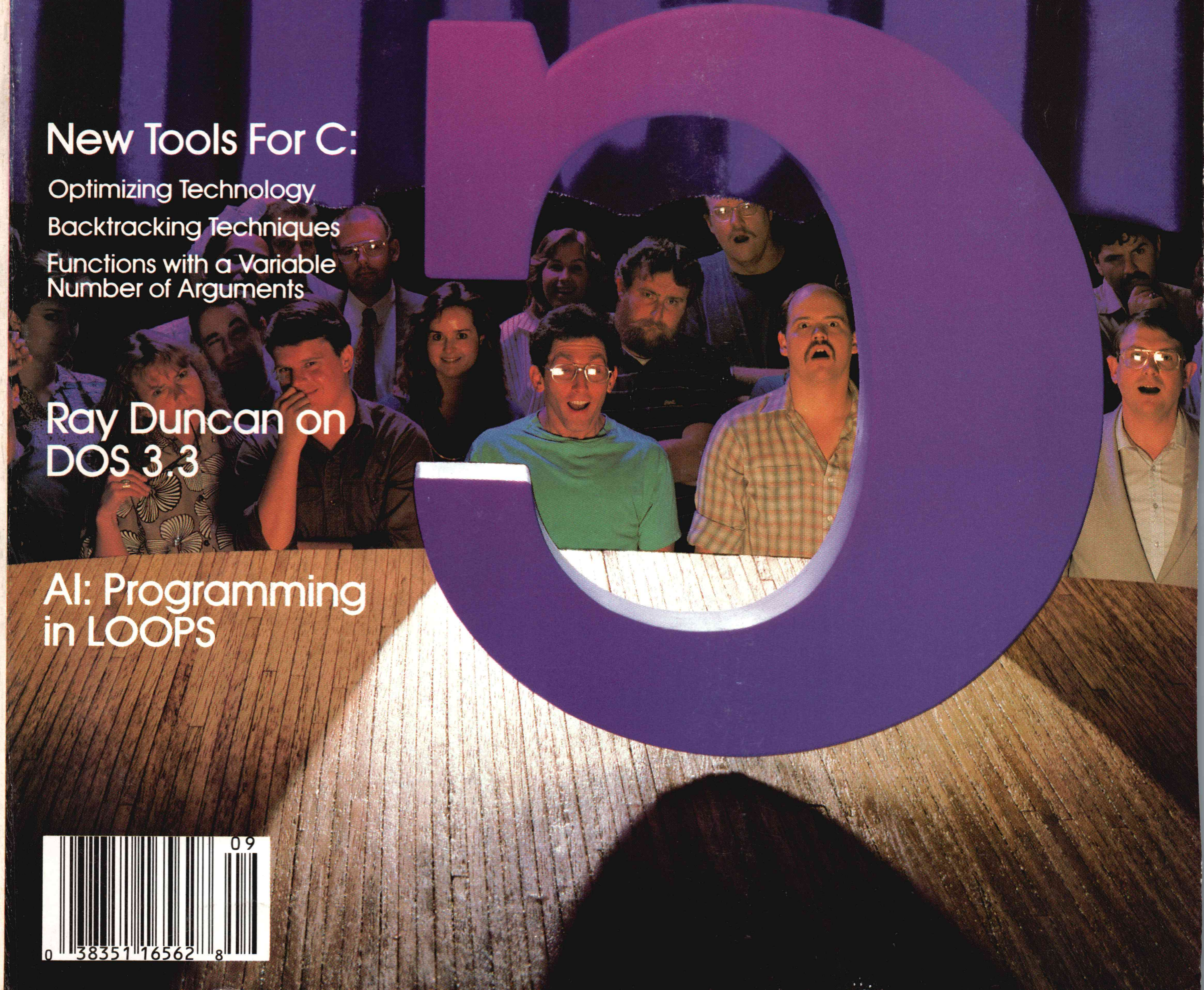
#130 AUGUST 1987

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

Unveiling ANSI C



New Tools For C:

- Optimizing Technology
- Backtracking Techniques
- Functions with a Variable Number of Arguments

Ray Duncan on DOS 3.3

AI: Programming in LOOPS

DR. DOBB'S JOURNAL

#130 AUGUST 1987

bo C: **NEW!**

erful optimizing

er ever

Sieve benchmark

	Turbo C	Microsoft® C
Compile time	*2.4*	13.51
Compile and link time	*4.1*	18.13
Execution time	*3.95*	5.93
Object code size	*239*	249
Execution size	*5748*	7136
Price	*$99.95*	$450.00

Benchmark run on an IBM PS/2 Model 60 using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51.

Technical Specifications

- ☑ Compiler: One-pass optimizing compiler generating linkable object modules. Included is Borland's high-performance Turbo Linker.™ The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).
- ☑ Interactive Editor: The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.
- ☑ Development Environment: A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pull-down menus and windows.
- ☑ Links with relocatable object modules created using Borland's Turbo Prolog into a single program.
- ☑ Inline assembly code.
- ☑ Loop optimizations.
- ☑ Register variables.
- ☑ ANSI C compatible.
- ☑ Start-up routine source code included.
- ☑ Both command line and integrated environment versions included.
- ☑ License to the source code for Run-time Library available.

Join more than 100,000 Turbo C enthusiasts. Get your copy of Turbo C today!

Minimum system requirements: All products run on IBM PC, XT, AT, PS/2, portable and true compatibles. PC-DOS (MS-DOS) 2.0 or later. 384K RAM minimum. Basic Telecom and Editor Toolboxes require 640K.

Borland International
4585 Scotts Valley Drive, Scotts Valley, CA 95066
Telephone: (408) 438-8400 Telex: 172373

Why more than 600,000 programmers worldwide are using Turbo Pascal today

The irresistible force behind Turbo Pascal's worldwide success is Borland's advanced technology. We created a compiler so fast, that Turbo Pascal® is now the worldwide standard. And there are more tools for Turbo Pascal than for any other development environment in the world.

You'll get everything you need from Turbo Pascal and its 5 Toolboxes

Turbo Pascal and Family are all you'll ever need to perfect programming in Pascal.

If you've never programmed in Pascal, you'll probably want to start with Turbo Pascal Tutor® 2.0, and as your expertise quickly grows, add Toolboxes like our

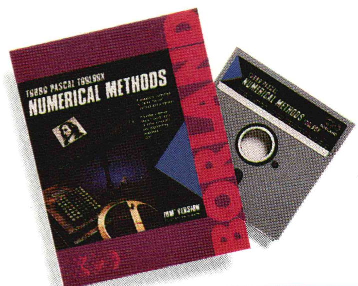
- Database Toolbox®
- Editor Toolbox®
- Graphix Toolbox®
- GameWorks®
  and our newest,
- Numerical Methods Toolbox™

And because Turbo Pascal is the established worldwide standard, 3rd party, independent non-Borland developers also offer an incredible array of programs for Turbo Pascal. ***Only $99.95!***

"Borland International's Turbo Pascal took the programming world by storm. A great compiler combined with a good editor at an astounding price, the package quickly came to be called, simply, Turbo—and has sold more than 500,000 copies.

Stephen Randy Davis, PC Magazine

Language deal of the century. *PC Magazine*"



For Scientists and Engineers: Turbo Pascal Numerical Methods Toolbox

The Numerical Methods Toolbox is a complete collection of Turbo Pascal routines and programs. Add it to your development system and you have the most comprehensive and powerful numerical analysis capabilities—at your fingertips!

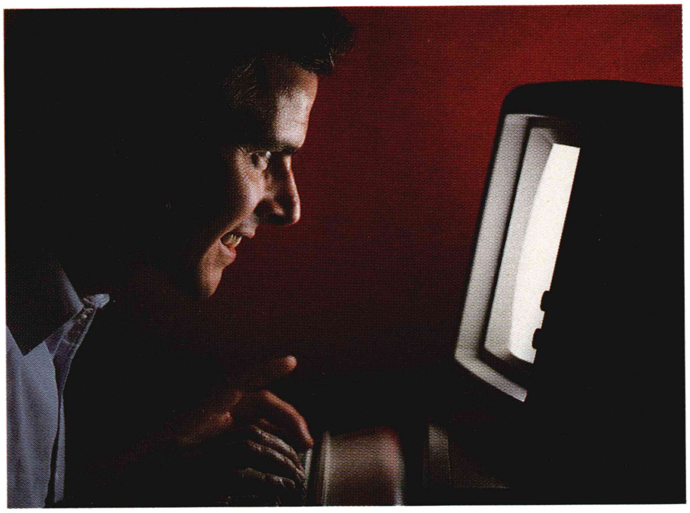
The Numerical Methods Toolbox is a state-of-the-art mathematical toolbox with these ten powerful features:

- ☑ Zeros of a function
- ☑ Interpolation
- ☑ Differentiation
- ☑ Integration
- ☑ Matrix Inversion
- ☑ Matrix Eigenvalues
- ☑ Differential Equations
- ☑ Least Squares
- ☑ Fourier Transforms
- ☑ Graphics

Each module comes with procedures that can be easily adapted to your own program. The Toolbox also comes complete with source code. So you have total control of your application.

Only $99.95!

Turbo C, Turbo Basic, Turbo Pascal and Turbo Prolog: technical excellence



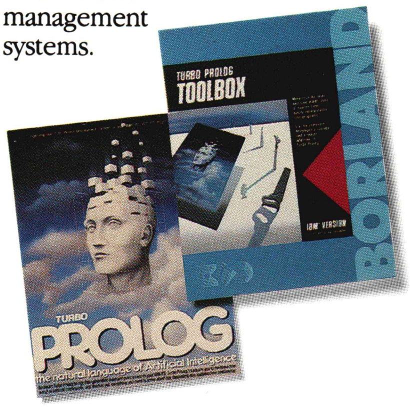
"Borland International's Turbo Pascal, Turbo Basic and Turbo Prolog automatically identify themselves, by virtue of their 'Turbo' forenames, as superior language products with a common programming environment. The appellation also means to many PC users a 'must have' language. To us Turbo C looks like a coup for Borland.

Garry Ray, PC Week"

BI-1131

Turbo Prolog: The Natural Language of Artificial Intelligence

Whether you're a first-time programmer or an experienced one, Turbo Prolog's natural implementation of Artificial Intelligence soon shows you how to build expert systems, natural language interfaces, customized knowledge bases and smart information management systems.



Turbo Prolog and Turbo C work hand-in-hand

Turbo Prolog® interfaces perfectly with Turbo C® because they're both designed to work with each other.

The Turbo Prolog/Turbo C combination means that you can now build powerful commercial applications using two of the most powerful languages available.

Turbo Prolog's development system includes:

- ☑ A complete Prolog compiler that is a variation of the Clocksin and Mellish Edinburgh standard Prolog.
- ☑ A full-screen interactive editor.
- ☑ Support for both graphic and text windows.
- ☑ All the tools that let you build your own expert systems and **AI** applications with unprecedented ease.

"An affordable, fast, and easy-to-use language that will delight the newcomer . . . You experienced Prolog hackers will likewise be delighted, if not astonished, by the features and performance of the Turbo Prolog development environment.

Turbo Prolog offers generally the fastest and most approachable implementation of that language.

Darryl Rubin, AI Expert"

How Turbo Prolog's new Toolbox adds 80 powerful tools and 8000 lines of source code

In keeping with Borland tradition, we've quickly added the new Turbo Prolog Toolbox™ to Turbo Prolog.

With 80 tools and 8000 lines of source code that can easily be incorporated into your own programs—and 40 sample programs that show you how to put these AI tools to work—the Turbo Prolog Toolbox is a highly intelligent, high-performance addition.
Only $99.95!

Turbo Prolog Toolbox features include:

- ☑ Business graphics generation: boxes, circles, ellipses, bar charts, pie charts, scaled graphics
- ☑ Complete communications package: supports XMODEM protocol
- ☑ File transfers from Reflex,® dBASE III,® 1-2-3,® Symphony®
- ☑ A unique parser generator: construct your own compiler or query language
- ☑ Sophisticated user-interface design tools
- ☑ Contains 40 example programs
- ☑ Easy-to-use screen editor: design your screen layout and I/O
- ☑ Calculated fields definition
- ☑ Over 8,000 lines of source code you can incorporate into your own programs

Tur
The most pov
comp

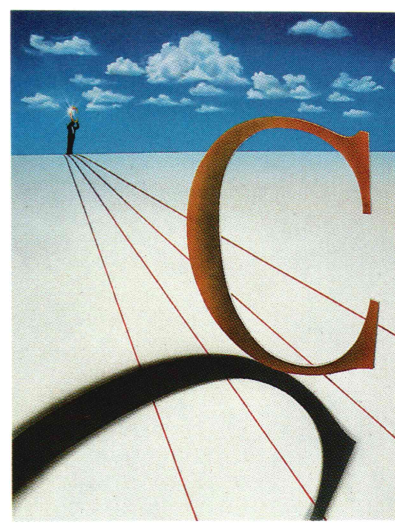
Our new Turbo C generates fast, tight, production-quality code at compilation speeds of more than 13,000 lines a minute!

It's the full-featured optimizing compiler everyone has been waiting for.

Switching to Turbo C, or starting with Turbo C, you win both ways

If you're already programming in C, switching to Turbo C will make you feel like you're riding a rocket instead of pedaling a bike.

If you're never programmed in C, starting with Turbo C gives you an instant edge. It's easy to learn, easy to use, and the most efficient C compiler at any price.
Only $99.95!



"Turbo C does look like What We've All Been Waiting For: a full-featured compiler that produces excellent code in an unbelievable hurry . . . moves into a class all its own among full-featured C compilers . . . Turbo C is indeed for the serious developer . . . One heck of a buy—at any price.

Michael Abrash, Programmer's Journal"

Turbo Basic introduces its powerful new Telecom, Editor and Database Toolboxes

NEW!

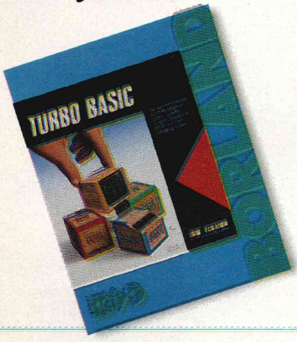
Turbo Basic® is the breakthrough you've been waiting for. The same power we brought to Pascal with Turbo Pascal has now been applied to BASIC with Turbo Basic.

Compatible with BASICA, Turbo Basic is the high-performance, high-speed BASIC you'd expect from Borland.

Basically, Turbo Basic is all you need

It's a complete development environment which includes an incredibly fast compiler, an interactive editor and a trace debugging system. It outperforms all its rivals, and because it's compatible with BASICA, you probably already know how to use it.

Includes a free MicroCalc™ spreadsheet complete with source code. ***Only $99.95!***



A technical look at Turbo Basic

- ☑ Full recursion supported
- ☑ Standard IEEE floating-point format
- ☑ Floating-point support, with full 8087 (math co-processor) integration. Software emulation if no 8087 present
- ☑ Program size limited only by available memory (no 64K limitation)
- ☑ VGA, CGA, and EGA support
- ☑ Access to local, static, and global variables
- ☑ Full integration of the compiler, editor, and executable program, with separate windows for editing, messages, tracing, and execution
- ☑ Compile, run-time, and I/O errors place you in the source code where error occurred
- ☑ New long integer (32-bit) data type
- ☑ Full 80-bit precision
- ☑ Pull-down menus
- ☑ Full window management

> " Borland has created the most powerful version of BASIC ever.
>
> *Ethan Winer, PC Magazine* "

NEW!

Telecom Toolbox is a complete communications package which takes advantage of the built-in communications capabilities of BASIC—use as is or modify.

- Pull-down menus and windows
- XMODEM support
- VT 100 terminal emulation
- Captures text to disk or printer
- PhoneBook file
- 300, 1200, 2400 baud support
- Supports script files
- Fast screen I/O
- Supports most of XTalk's command set
- Manual dial and redial options

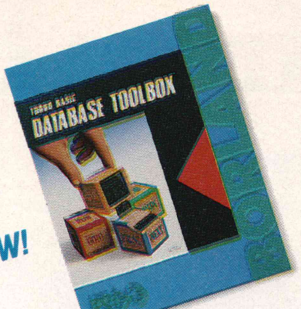
Use Telecom Toolbox to embed communications capabilities into your own programs and/or build your own communications package. Source code included for all Toolbox code and sample programs. ***Only $99.95!***

For the dealer nearest you or to order by phone call

(800) 255-8008

in CA (800) 742-1133 in Canada (800) 237-1136

BORLAND
INTERNATIONAL



NEW!

Database Toolbox means that you don't have to reinvent the wheel each time you write new Turbo Basic database programs.

- ☑ "Trainer" shows you how B+ trees work. (Simply key in sample records and you'll see your index being built.)
- ☑ Turbo Access instantly locates, inserts or deletes records in a database—using B+ trees.
- ☑ Turbo Sort sorts data on single items or on multiple keys and features virtual memory management for sorting large data files.

Source code included.

Only $99.95!

NEW!

Editor Toolbox is all you need to build your own text editor or word processor. Includes source code for two sample editors.

First Editor is a complete editor ready to include in your programs, complete with windows, block commands and memory-mapped screen routines.

MicroStar™ is a full-blown text editor with a complete pull-down menu user interface, and gives you

- Wordwrap
- Undo last change
- Auto-Indent
- Find and Find/Replace with options
- Set left/right margins
- Block mark, move and copy
- Tab, insert, overstrike modes, line center etc.

Includes source code.

Only $99.95!

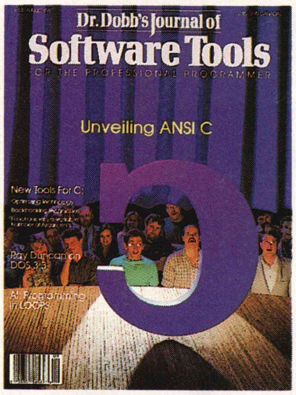
BI-1131

CIRCLE 161 ON READER SERVICE CARD

AUGUST 1987 VOLUME 12, ISSUE 8

CONTENTS

ARTICLES



About the Cover
The arrival of ANSI C is something many of us are looking forward to, but it may be that not all the surprises are pleasant ones.

COLUMNS

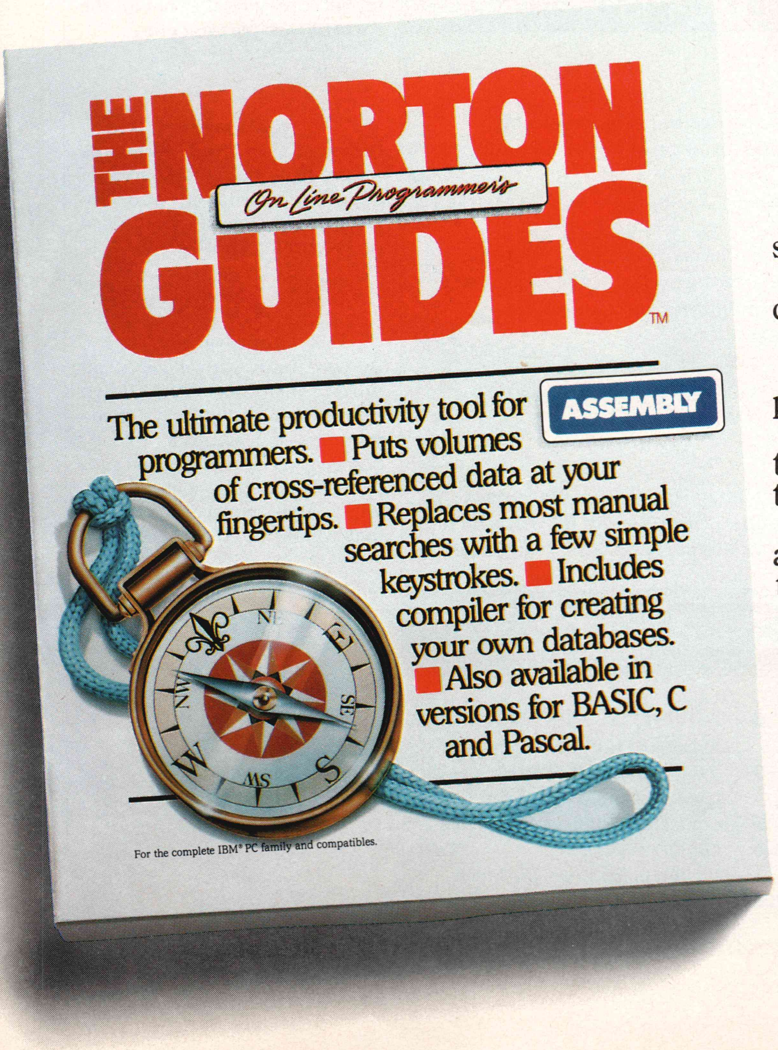
This Issue
Our focus on the C language ranges from the cutting edge of compiler technology and language extensions to traditional programming utilities.

Next Issue
So you think you've got it all wired just because you've memorized the first three volumes of Knuth's *Art of Computer Programming?* Be sure to check out next month's issue, which will explore algorithms that even Knuth hasn't heard of.

FORUM

PROGRAMMER'S SERVICES

Peter Norton
new programmi
who hat



Nobody ever said programming PCs was supposed to be easy.

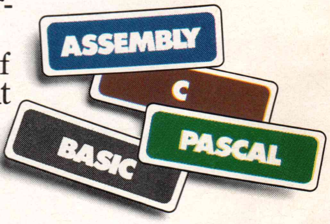
But does it have to be tedious and time-consuming, too?

Not any more.

Not since the arrival of the remarkable new program on the left.

Which is designed to save you most of the time you're currently spending searching through the books and manuals on the shelf above.

The Norton On-Line Programmer's Guides™ are a quartet of pop-up reference packages that do the same things in four different languages.



Each package consists of two parts: A memory-resident instant access program. And a comprehensive, cross-referenced database crammed with just about everything you need to know to program in your favorite language.

And when we say everything, we mean everything.

Everything from information about language

Designed for the IBM® PC, PC-AT and DOS compatibles. Available at most software

announces a
ng tool for people
e manual labor.

syntax to a variety of tables, including ASCII characters, line drawing characters, error messages, memory usage maps, important data structures and more.

How much more?

Well, the databases for BASIC, C and Pascal give you detailed listings of all built-in and library functions.

While the Assembly database delivers a complete collection of DOS service calls, interrupts and ROM BIOS routines.

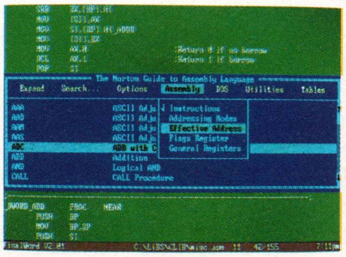
You can, of course, find most of this information in the books and manuals on our shelf.

But Peter Norton—who's written a few books himself—figured you'd rather have it on your screen.

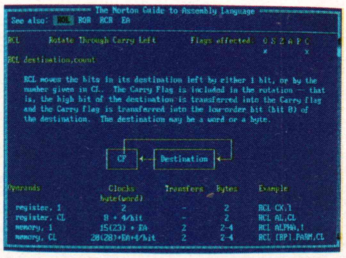
In seconds.

In full-screen or moveable half-screen mode.

Popping up right next to your work. Right where you need it.



A Guides reference summary screen (shown in blue) pops up on top of the program you're working on (shown in green).



Summary data expands on command into extensive detail. And you can select from a wide variety of information.

This, you're probably thinking, is precisely the kind of thinking that produced the classic Norton Utilities.™

And you're right.

But even Peter Norton can't think of everything.

Which is why there's a built-in compiler for creating databases of your own.

And why all Guides databases are compatible with the instant access program in your original package.

So you can add more languages without spending a lot more money.

To get more information, call your dealer or Peter Norton Computing.

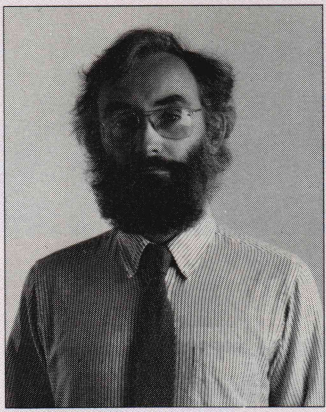
And ask for some guidance.

Peter Norton
COMPUTING

dealers, or direct from Peter Norton Computing, Inc., 2210 Wilshire Blvd., #186, Santa Monica, CA 90403. 213-453-2361, Fax 213-453-6398, MCI Mail: PNCI ©1987 Peter Norton Computing

CIRCLE 87 ON READER SERVICE CARD

EDITORIAL



DDJ has been interested—make that involved—in the spread of the C programming language since we published the source code for Ron Cain's Small C compiler back in 1980. At that time there was no commercial C compiler for a personal computer. In the intervening years, C has grown to its present position as the preeminent language for software development on and for personal computers.

Critics of the language can say that, like BASIC before it, C is not well structured, is hard to read, and is hard to maintain. They can't say it is unpopular. C's myriad supporters will tell you that it provides all the low-level access they normally need; that C code, properly written, is well structured and is not hard for a C programmer to read or to maintain; that the language allows both structure and freedom; that it is an empowering tool. The supporters' view has prevailed and has led to C's currently being available in any environment you care to name, with hordes of compiler vendors riding the wave.

Today an ANSI standard for C has all but been established; you'll find a discussion of the draft standard in this issue.

So the C programming language is uncommonly popular among serious developers, is available in almost every conceivable programming environment, and is on the verge of standardization. But the waters never settle except to be stirred anew.

Up through this common ground of popularity, ubiquity, and standardization, a watershed is arising, with C product designs flowing in one of two directions: toward deeper levels of optimization on the one side and toward ease of learning, ease of use, and speed of the development cycle on the other. Thus we see a growing pool of compilers from Datalight and Microsoft, for example, that promise optimization technology heretofor only available on minicomputers and mainframes; and we also see products like Borland's Turbo C, Microsoft's Quick C, and Mark Williams's Let's C that promise to extend the model of Turbo Pascal or interpreted BASIC to C. It is not clear whether this parting of the Cs is caused by the Moses of market dynamics or by some deep tectonics of inherently incompatible programmer needs. But the decision to go with the flow and to learn/buy/use C now leads to another decision as to which flow with which to go.

And tomorrow? The C programming language seems so widespread that it is more likely to adapt than to be replaced. It seems reasonable that we will sooner, rather than later, see some of the following: 4GLs built on top of C, a variety of preprocessors and user interfaces, support for different compilation models, and extensions of the C packages sold in the direction of rich sets of version control tools.

In the future C will change into something rich and strange.

Michael Swaine

Michael Swaine
editor-in-chief

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Editorial
Editor-in-Chief *Michael Swaine*
Editor *Tyler Sperry*
Managing Editor *Vince Leone*
Assistant Editors *Sara Noah Ruddy*, *Levi Thomas*
Technical Editors *Allen Holub*, *Richard Relph*
Contributing Editors *Ray Duncan*, *Michael Ham*, *Namir Shammas*, *Ernest R. Tello*
Copy Editor *Rhoda Simmons*
Production
Production Manager *Bob Wynne*
Art Director *Michael Hollister*
Assoc. Art Director *Joe Sikoryak*
Technical Illustrator *Frank Pollifrone*
Cover Photographer *Michael Carr*
Circulation
Circulation Director *Maureen Kaminski*
Book Marketing Mgr. *Jane Sharninghouse*
Circulation Coordinator *Kathleen Shay*
Administration
Finance Director *Kate Wheat*
Business Manager *Betty Trickett*
Accounts Payable Supv. *Mayda Lopez-Quintana*
Accts. Receivable Supv. *Laura Di Lazzaro*
Advertising Director
Ferris Ferdon (415) 366-3600
Account Managers

see page 113

Promotions/Srvcs. Mgr. *Anna Kittleson*
Advertising Coordinator *Charles Shively*
Associate Publisher
Michael Swaine
Assistant *Sara Noah Ruddy*

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points. *DDJ* is published under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

Article Submissions: Send manuscripts and disk (with article and listings) to the Editor.

***DDJ* on CompuServe:** Type GO DDJ

Address Correction Requested: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 0888-3076**

Customer Service: For subscription problems call: outside CA (800) 321-3333; in CA (619) 485-9623 or 566-6947. For book/software order problems call (415) 366-3600.

Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $27 per year airmail or $10 per year surface. All other countries add $27 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX: 620430 (WUI).

M&T Publishing Inc.
Chairman of the Board *Otmar Weber*
Director *C. F. von Quadt*
President and Publisher *Laird Foshay*

Our thanks to NASA for supplying this computer enhanced ultraviolet photo taken by Skylab IV of a solar prominence reaching out 350,000 miles above the sun's surface

Genius Begins With A Great Idea ...

But The Idea Is Just The Beginning

What follows is the time consuming task of giving form and function to the idea.

That's why we concentrate on building into our software development systems functions and features that help you develop your software ideas in less time and with less effort.

We've started 1987 by releasing new versions of our MS-DOS, Macintosh, Amiga, ROM, and Apple // C development systems. Each system is packed with new features, impressive performance, and a little bit more genius.

FREE 2-DAY DELIVERY!

Aztec C86 4.1

New PC/MS-DOS • CP/M-86 • ROM

Superior performance, a powerful new array of features and utilities, and pricing that is unmatched make the new Aztec C86 the first choice of serious software developers.

Aztec C86-p Professional System $199
• optimized C with near, far, huge, small, and large memory + Inline assembler + Inline 8087/80287 + ANSI support + Fast Float (32 bit) + optimization options • Manx Aztec 8086/80x86 macro assembler • Aztec overlay linker (large/small model) • source level debugger • object librarian • 3.x file sharing & locking • comprehensive libraries of UNIX, DOS, Screen, Graphics, and special run time routines.

Aztec C86-d Developer System $299
• includes all of Aztec C86-p • Unix utilities make, diff, grep • vi editor • 6+ memory models • Profiler.

Aztec C86-c Commercial System. $499
• includes all of Aztec C86-d • Source for library routines • ROM Support • CP/M-86 support • One year of updates.

Aztec C86 Third Party Software

A large array of support software is available for Aztec C86. Call or write for information. The following is a list of the most requested products: Essential Graphics • C Essentials • C Utility Library • Greenleaf Com. • Greenleaf General • Halo • Panel • PC-lint • PforCe • Pre-C • Windows for C • Windows for Data C terp • db Vista • Phact • Plink86Plus • C-tree.

CP/M • TRS-80 • 8080/Z80 ROM

C compiler, 8080/Z80 assembler, linker, librarian, UNIX libraries, and specialized utilities.

Aztec C II-c (CP/M-80 & ROM). $349
Aztec CII-d (CP/M-80) $199
Aztec C80 (TRS-80 3&4) $199

Aztec C68k/Am 3.4

New Amiga Release

Amiga user groups across the USA voted Aztec C68k/Am release 3.3 the best Software Development System for the Amiga. Release 3.4 is more impressive.

Aztec C68k/Am-p Professional $199
A price/feature/performance miracle. System includes: optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Amiga libraries • examples.

Aztec C68k/Am-d Developer $299
The best of Manx, Amiga, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep and vi.

Aztec C68k/Am-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

FREE 2-DAY DELIVERY!

Aztec C68k/Mac 3.4

New Macintosh Release

For code quality, reliability, and solid professional features, Aztec C for the Macintosh is unbeatable. This new release includes features and functions not found in any other Macintosh C development system.

Aztec C68k/Mac-p Professional $199
• optimized C • 68000/680x0 assembler • 68881 support • overlay linker • UNIX and Macintosh libraries • examples.

Aztec C68k/Mac-d Developer. $299
The best of Manx, Macintosh, and UNIX. System includes: all of Aztec C68k/Am-p • the Unix utilities make, diff, grep • vi editor.

Aztec C68k/Mac-c Commercial $499
Aztec C68k/Am-d plus source for the libraries and one year of updates.

Aztec C65

New ProDOS Release

Aztec C65 is the only commercial quality C compiler for the Apple II. Aztec C65 includes C compiler, 6502/65C02 assembler, linker, library utility, UNIX libraries, special purpose libraries, shell development environment, and more. An impressive system.

Aztec C65-c Commercial $299
• runs under ProDOS • code for ProDOS or DOS 3.3

Aztec C65-d Developer. $199
• runs under DOS 3.3 • code for DOS 3.3

Aztec ROM Systems

6502/65C02 • 8080/Z80 • 8086/80x86 • 680x0

An IBM or Macintosh is not only a less expensive way to develop ROM code, it's better. Targets include the 6502/65C02, 8080/Z80, 8086/80x86, and 680x0.

Aztec C has an excellent reputation for producing compact high performance code. Our systems for under $1,000 outperform systems priced at over $10,000.

Initial Host Plus Target $750
Additional Targets $500
ROM Support Package $500

Vax, Sun, PDP-11 ROM HOSTS

Call for information on Vax, PDP-11, Sun and other host environments.

C' Prime

PC/MS-DOS • Macintosh
Apple II • TRS-80 • CP/M

These C development systems are unbeatable for the price. They are earlier versions of Aztec C that originally sold for as much as $500. Each system includes C compiler, assembler, linker, librarian, UNIX routines, and more. Special discounts are available for use as course material.

C' Prime $75

Aztec Cross Development Systems

Most Aztec C systems are available as cross development systems. Hosts include: PC/MS-DOS, Macintosh, CP/M, Vax, PDP-11, Sun, and others. Call for information and pricing.

How To Become An Aztec C User

To become a user call 800-221-0440. From NJ or international locations call 201-542-2121. Telex: 4995812 or FAX: 201-542-8386. C.O.D., VISA, MasterCard, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Aztec C is available directly from Manx and from technically oriented computer and software stores. Aztec Systems bought directly from Manx have a 30 day satisfaction guarantee.

Most systems are upgradable by paying the difference in price plus $10. Site licenses, OEM, educational, and multiple copy discounts are available.

MANX

To order or for more information call today.

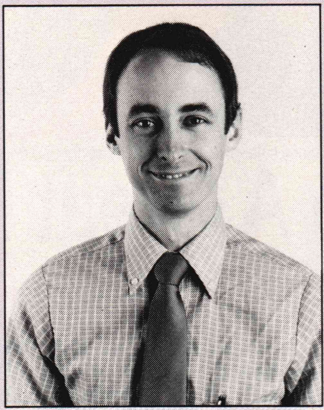
1-800-221-0440

In NJ or international call (201) 542-2121 • TELEX: 4995812

CIRCLE 108 ON READER SERVICE CARD

Manx Software Systems
1 Industrial Way, Eatontown, NJ 07724

RUNNING LIGHT



Last month I offered the imposter theory to explain that fellow on the right and his necktie. The fact that neckties cut off the vital flow of oxygen to the brain has always encouraged me to stay as far away from them as possible. Thus my doppelganger theory for the photo of the new editor at *DDJ*.

This month, however, you'll notice we have virtually the same photo as last month. Doesn't this seem odd to you? I mean, this guy's been wearing the very same shirt and tie for a solid month now, and he's still smiling? There's definitely something strange going on here.

I thought about it quite a bit, and I have a possible explanation. My theory is that the guy is a robot, an audioanimatronic device like Disneyland's Mr. Lincoln. Once a month they wheel him out, plug him into the wall so his eyes will light up, take the picture, and then shove him back in the closet. It's a fantastic theory, of course, and there's probably nothing to it. Still.... What if someone was replacing people in the computer community with robots? How many replacements would it take before we noticed something was amiss?

This robot hypothesis goes a long way toward explaining some of the strange things you see in computer magazines today. For example: how is it that someone could write a positively glowing "preview" of Borland's Turbo C months before the program had shipped and when the only information available was a pitch from a a company representative and a short demonstration of the program? How could a responsible journalist do something like that? With my theory, the answer becomes obvious: a real journalist wouldn't act that way, but a robot could easily be programmed for the job. Just set some global variables (such as *PRAISE = Max; CONTENT = Min*) and there you have it. No more troubles with freelance writers, no more deadline hassles, and no more unhappy advertisers. What could be better for a magazine?

Well, despite all those advantages, the folks here at *DDJ* are still operating with the traditional organic processors. Bit-slice editors and columnists would be faster, but they'd lack the experience and judgement that makes a magazine worth reading. So though this is our annual C issue, I have to confess we don't have a rush review of a new compiler. In the near future, we'll no doubt have some interesting things to say about Turbo C (and Microsoft's Quick C) but first we have to do our homework and actually spend some time evaluating the products.

One last thing before I go. If you're a hotshot on either operating systems or 68000 programming, give me a call at (415) 366-3600 and sell me on a brilliant article idea. The December and January issues are coming up fast, and I still haven't filled all the pages.

Tyler Sperry
editor

ARCHIVES

Ten Years ago in *DDJ*

"The neologism 'modem' stands for modular-demodulator. The need for modems arose when somebody wanted to send a digital signal to New Medford (for example) and noticed that there was a telephone line going to New Medford. But the telephone lines only carried audio. So this somebody made a device that turned a 0 into a 1070 Hz tone and a 1 into a 1270 Hz tone. This was a modulator. At the other end (probably in New Medford) a device was built that translated a 1070 Hz tone into a 0 and a 1270 Hz tone into a 1. This was a demodulator. Very much like a cassette interface."—*"Sour Notes on a Pennywhistle," Jef Raskin,* DDJ, *August 1977.*

"Dear Editor Persons: Do you think this is too drawn out to have the desired effect? (That is, wide eyes with dilated pupils, soft blue glow around the head (caused by surge in the Force), disorientation, repeated pronouncements such as 'wow' and 'fa-far-faarrout'.)

"On the other hand, if it were shortened to remove the fun and games, would it be appreciated by people who hadn't thought of the idea already?

"By the way, is it original? I wrote this between 12 and 2 in the morning . . ."—*letter that accompanied manuscript "On the Effects of Filling Cavities Within the Fillings of Cavities Within . . .," Steve Whitham,* DDJ, *August 1977.*

C Notes

"Alexander Graham Bell invented the tin ear, so maybe it's not surprising that Bell Lab's best software should have names like UNIX and C."—*"Binary Trees with Tiny-C," Les Hancock,* DDJ, *June/July 1979.*

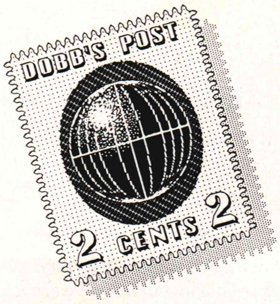
"C is a disease. When I see people writing spreadsheets in C, I think, 'They're out of their minds.' It was designed to write operating systems. Modula-2 is good for that [writing spreadsheets]. We'll do a C. We'll do a C because everyone wants a C. But in Europe C is seen as an American disease, and here people are trying to spread it."—*Philippe Kahn, quoted in The Software Designer,* DDJ, *October 1984.*

DDJ in a Nutshell

"*Dr. Dobb's Journal* began as a forum for sharing information and ideas about programming and computers. It continues to be a place to present new languages, utilities, tools, applications, algorithms, discoveries, and techniques to the microcomputing community. Our authors primarily come from within our readership, and it is this reader involvement that has sustained and guided *DDJ* throughout the years."—*Editorial, Reynold Wiggins,* DDJ, *April 1984.*

Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia
Running Light Without Overbyte

LETTERS



Clear, Compact Code

Dear *DDJ*,

While reading "Dimensional Data Types" by Do-While Jones (May 1987), I sat stunned in disbelief. He seems to miss the boat on virtually every point he tries to make.

"Compact code is no longer necessary or desirable." Several responses to this absurd statement immediately come to mind. But, as this is a family publication, just ask Microsoft about its Windows project. Just about every compiler advertised in your magazine boasts about its compact code generation. Compact code must be important to somebody. A wise engineer once told me, "If software ignores the hardware, you can be sure that the hardware will ignore the software."

"The big money is now starting to go to people who can write clear code." Mr. Jones seems to imply that complex code and clear (that is, properly documented) code are mutually exclusive. Again this is absurd. Some of the prettiest and cleanest code I have ever seen was a telecommunications system written in 8080 assembly language. I can also just about guarantee that the assembly-language code would be smaller and faster than the equivalent written in Ada (if an Ada could be found that ran on a small system). Ada, Pascal, and Modula-2 are not synonymous with clean and easy to maintain code. Some of the "dirtiest" production code I have worked on has been in Pascal. Another wise programmer once said, "A real programmer can write FORTRAN in any language."

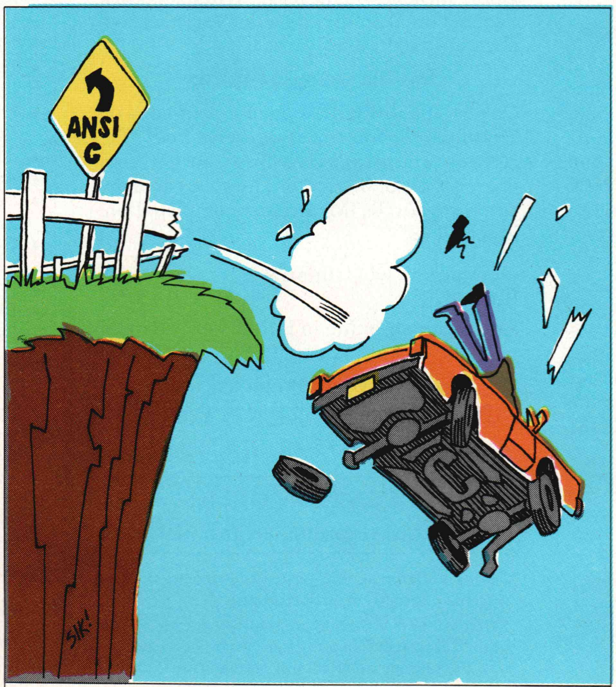
"What makes this program difficult to validate and maintain is the cryptic number *335,300,000*." Although I agree with Mr. Jones' statement, the way he made the program "uncryptic" was to add two pages of unit definitions so that the constant *c* could be defined and then divided by 2. It seems to me that, with the addition of one constant declaration and a comment line, the "bad" program could have been much easier to read than wading through the "good" (that is, longer) code.

"The overhead isn't as bad as it looks." I guess that, in a large, academic, mainframe Ada environment in which everything is slow, compile/link/load/execution times are a mere nit. I wish my customers were as forgiving. To use Mr. Jones' analogy of bank computers balancing checkbooks, I wonder which system bank presidents would want—a tight software system that gives them answers faster or an inefficient one that takes longer but has "cleaner" code? I guarantee that they don't care what the code looks like but that they do care how quickly the machines can give them information.

"Programmers should no longer waste time combining constants because the compiler should do it anyway." Do most programmers know what code their compiler actually generates? I doubt it. Do all compilers do this because Mr. Jones thinks they should? I doubt it.

Contrary to popular belief, performance still matters. CPU time is not infinite. Memory is not unlimited (virtual memory included). In the academic world unpleasantries such as the hardware can be ignored. In the real world, they cannot.

Eric Lundquist
Centurion Dealers Computer Corp.
402 West Bethany
Allen, TX 75002



FUNCTION PROTOTYPES:
Don't leave home without them.

Programmer's Hell

Dear *DDJ*,

"Factoring in Forth" by Michael Ham (October 1986) makes almost every mistake in the book. Although the author waxes eloquent about factoring, he gives listings that illustrate the three cardinal sins of Forth.

Although it is common to recommend short Forth definitions, you should not lose sight of two of the original aims of the language: compactness and speed. Each time you decompose a definition, you pay a penalty in time and space. Therefore, factoring is worthwhile only if one of the following two conditions is met: some of the components may be useful in other constructions, or the components contain some genuine ideas that may well be separated so that you can analyze their implementation. Listing Six in the article provides an ideal bad example—the only term used later is *@BIT*, which is factored using four new definitions. You could, however, define it more efficiently using only standard terms:

```
: @BIT SWAP 8 /MOD ROT +
C@ SWAP 0 DO 2/ LOOP 1 AND
NEG ;
```

(I use a *DO* so that *n n DO* does nothing.) Whether you want to introduce:

```
: AIM SWAP 8 /MOD ROT + '
```

depends on how often you intend to use this, but the other three new terms are useless; one—*S>B*—is a synonym for *0<>*. Who needs it?

Mr. Ham likes to make definitions using *CREATE* and then storing. He should use *VARIABLE* and *ALLOT*. This is not a pecadillo. In debugging Forth you need to know what words do. You can consider the dictionary as being divided into primitives and high-level terms. Primitives are machine-code definitions; there has to be some way of flagging them. Then you need a disassembler to see what is there. The high-level terms are those defined in terms of primitives and other high-level terms. On the first level, you have two species—defining words using *;CODE* and defining words using *DOES>*. Those that use *;CODE* should be few in number so they can be traced. Typical examples are *:* and *VARIABLE*. You recognize a colon definition by looking at the contents of its CFA. For *DOES>* there is no problem because the contents of the CFA signal a *DOES>* definition and the first word of the parameter field points to the address where the procedure is given. Instead of the definition of *BITS* given in the listing, Mr. Ham should have used:

```
VARIABLE BITS 6 ALLOT
```

This tells anyone that *BITS* is a storage word because its CFA will contain the address of the *;CODE* procedure for *VARIABLE*.

Finally, Listing Five contains a disaster—*VECTOR:*. This is a defining term that creates words requiring an input, and wrong input here could destroy a disk. When you do any serious programming, you make lots of mistakes before you get it right. The most serious problem with Forth is that it lets you do anything you want, including making catastrophic mistakes. The most common source of such errors is storing data in the wrong place because you may overwrite code. At best you won't encounter the code you have destroyed. At second best you get a crash. At worst you may have created new code that makes sense but writes nonsense to a disk directory. I've done this too often.

Mr. Ham goes one better with his execution arrays—he seems to think the big issue is a matter of changing the name *VECTOR:* to *EMPOWER*. He lists:

```
4 DO-OPTION ( unpredictable results )
```

where *DO-OPTION* is defined using *VECTOR:*. It would be nice if the results were unpredictable, but I'll tell you what will happen: the CPU will start reading a location in memory as if it were instruction code. If you're lucky, it reaches nonsense code before much has happened and a mere crash results. If you are unlucky, it will start executing something that is indeed unpredictable, but it may be a disk-write or it may be a change in DOS. The chances of wrecking a disk are not negligible, even if small (destroying a hard disk even once a year is not worth it).

The fact is that a definition such as *VECTOR:* should send its perpetrator to Programmer's Hell. (The *CASE* statement could do what Mr. Ham wants with complete safety and no new defining terms.) Regardless of the language being used, the first duty of any programmer designing an interactive program is to anticipate wrong input from users. This is the hardest part of programming. At the least, users pressing the wrong key should not cause a catastrophe. What are called "user friendly" programs generally have an easy time—they are genuinely "programmer friendly" because the users have limited options. Command-driven programs, especially those written in Forth, let users make many mistakes. Some of these are their responsibility: if they have to type *ENDEDIT* to leave a word processor, it's not the programmer's duty to worry about whether they wanted to save first. In contrast, if typing *4* instead of *3* wrecks a disk instead of taking you to DOS as you intended, you may be tempted to enter the capital punishment debate on the wrong side.

Carl Herz
McGill University
Mathematics & Statistics
805 Sherbrooke West
Montreal, QB
Canada H3A 2K6

Michael Ham replies:
I agree with Carl's two conditions regarding when a name is needed but add also a third: factor when a name will make the code more readable. Compactness and speed are, of course, desirable, and if the additional definition has a serious negative impact on these, by all means rethink the factoring. But I find that in looking at code long after I wrote it, I am greatly helped by names. *S>B* (single to Boolean, named by analogy with *S>D*) helps me understand what is happening in a way that *0<>* does not. Another example: I typically use the definition *0 CONSTANT US>D* to be able to use the informative label *US>D* (unsigned single converted to double) in my source code rather than a mysterious *0*. Similarly, *AIM* was easier for me to read and remember than were the words in its definition. In the application I wrote, the speed and size penalties were negligible.

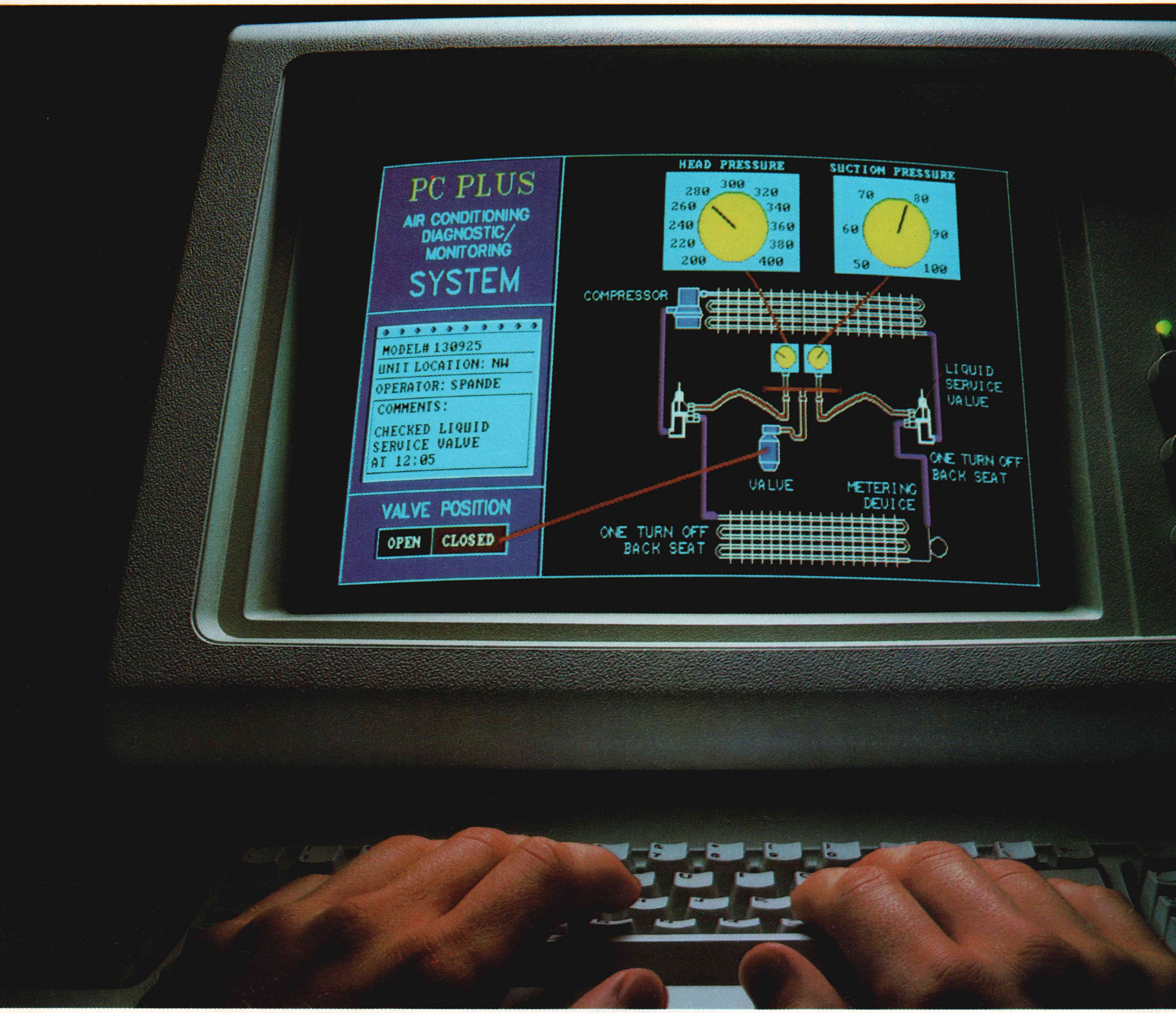
Of the definitions *+BIT*, *-BIT*, *@BIT*, and *~BIT*, it is true, as Carl points out, that I used only *@BIT* in the column. I should have made it clearer that the other words were offered for the reader's toolbox, should he or she sometime want to set, unset, or toggle a bit in addition to fetching it.

I appreciate Carl's suggestions of using *VARIABLE* when creating arrays of constants. The values within the variable can then be set with *C!*. (Values beyond the variable can still be stored with *C,* and so the *ALLOT* is not needed in this case.) His approach certainly works, and I recommend it to those who make heavy use of disassemblers.

I also agree with Carl's warnings about the disasters that can ensue from executing random locations in memory. I have never had the misfortunes he describes, but I can understand that they are possible. I am

(continued on page 140)

Texas Instruments has system developers need.



"Personal Consultant™ Plus . . . offers a very fine expert system development and delivery tool that already has a proven record with end-users."

— *Susan Shepard, AI Expert*

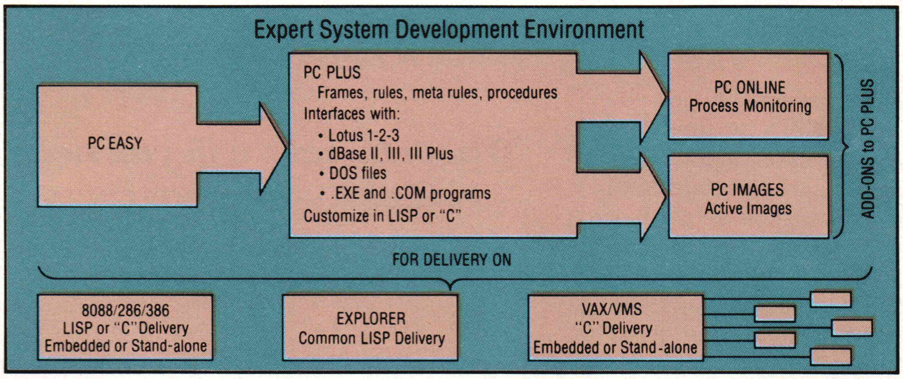
Personal Consultant Plus 3.0 Standard Features
- Frames, rules, meta rules and procedures
- Forward/backward chaining
- Confidence factors
- Regression testing and rule tracing
- End-user explanation facilities
- Graphics image capture and display
- Interfaces to dBase™, Lotus 1-2-3™, DOS files, .EXE or .COM programs, "C"
- Complete LISP development environment
- 2-megabyte expanded/extended memory support
- Mouse support
- Context sensitive help
- "Getting Started" tutorial-style manual

Personal Consultant Images
- Optional add-on package to PC Plus (3.0)
- Allows integration of "active images" into

what serious expert Power tools.

Among all the expert system development tools available for personal computers today, none deliver the power and flexibility of TI's Personal Consultant series.

Personal Consultant Easy is ideal for getting started, and is upwardly compatible with the higher functionality of PC Plus. For experienced developers, Personal Consultant Plus and its optional add-on enhancements, Online and Images, were designed to help solve a broader range of complex problems.



Personal Consultant Plus. Full power for an affordable price.
At $2,950, PC Plus has proven to be one of the richest and most flexible problem-solving tools available for the development of complex knowledge-based systems. Designed to take advantage of today's more powerful 286/386 DOS-based computers, or TI's Explorer™ Symbolic Processing System, the new 3.0 version of PC Plus provides powerful standard features and a continuing growth path with the addition of either PC Images or PC Online, or both.

Personal Consultant Images. Picture an expert system with interactive graphics.
At $495, PC Images enables developers to create knowledge-based applications that incorporate complex graphical "active images." User-interactive dials, gauges, forms and selection images provide a more exciting visual data input and output style.

Personal Consultant Online. The expert system as part of the process.
At $995, PC Online allows the developer to design expert systems which interact directly with process data, as opposed to input from a human operator. Designed for intelligent process monitoring applications, this optional package helps deliver expertise that is "online all the time."

Application delivery as flexible as the tools themselves.
Delivery can be in LISP for flexibility, or "C"* for maximum speed and portability. Our "C" options support either stand-alone or "embedded" knowledge bases. Options are available for DOS-based PCs, TI's Explorer, and DEC's VAX™ line of multi-user minis running under VMS™.

"Texas Instruments has done more than any other company to educate people about AI, to popularize it, and to make useful AI tools available at reasonable prices."
— Jim Seymour, PC Magazine.

Technical support, training courses and Knowledge Engineering Services are available for the Personal Consultant products. If you have a question about any of our expert system power tools, we have the answer.

Pick up the phone and gain a powerful advantage.
Call **1-800-527-3500** for technical overviews of our products and a PC Plus case histories brochure which details how our power tools are being put to work today.

knowledge bases
- Interactive dials, gauges, forms and selection images
- Multiple images can be combined on same screen
- "Getting Started" tutorial-style manual

Personal Consultant Online
- Optional add-on package for PC Plus (3.0)
- ONLINE expert systems that interact directly with process data
- Multiple interfaces to data acquisition and analysis programs
- Knowledge base synchronization with process data
- Functions for historical and predicted trends
- Special user interface/reporting capabilities
- "Getting Started" tutorial-style manual

TEXAS INSTRUMENTS

Preparing for ANSI C

by Richard Relph

What changes will you need to make your programs work with an ANSI compiler?

As most of you know by now, there is an effort afoot by the American National Standards Institute (ANSI) to standardize the C language. Some articles about the standard, most of them pretty general and glossy, have appeared in various magazines. This article is intended to be specific and useful. I hope to give you an idea of how the standard may be different from the C you're used to, how it will affect the code you have already written, and how you can minimize your adjustments to the new compilers by taking certain actions now.

Make no mistake, ANSI C is coming. Microsoft, AT&T, IBM, and many other companies are actively participating in the standard process by sending representatives (usually compiler writers) to the ANSI C committee meetings, which are held once every three months. Many of these companies, including AT&T, have announced that they will provide ANSI-conforming compilers. Although the standard will probably not be formally accepted until early 1988, nearly all the issues have been settled. The current working document (called the draft) will probably be changed very little before it becomes "the" standard.

I've written this article in a format similar to the organization of a normal C source file. After discussing a few general items from the standard, I'll discuss preprocessor directives, module-scope declarations, function definitions, function bodies, and the final topic will be libraries.

Richard Relph, 846 Salt Lake Dr., San Jose, CA 95133. Richard is a software and hardware consultant. He has written compilers and embedded systems.

General Items

To get around limitations in certain standard character sets, the committee has introduced "trigraphs." Trigraphs are three-character sequences, beginning with *??*, that the compiler replaces with a character that cannot be entered directly into a given machine—for example, *??=* is used for *#*, *??!* for ¦, and so on. I doubt that trigraphs will see much use in the U.S., but in certain countries they'll make the difference between only admiring C and being able to use it. Note that trigraphs are replaced even inside strings.

One problem area the standard addresses is the name space of programs. Which names are permissible for programmers to use, which are reserved by the compiler, and which are used by the library has always been an open issue.

The new standard includes a rule that says all names beginning with _ belong to the implementation and should not be used by any program. Also, any identifiers specified in the standard as keywords, functions, or macros are reserved, regardless of whether or not the header file that defines the function or macro is included. All other names belong to the programmer.

In order to allow programmers to write programs and be able to compile them on any machine that has an ANSI-conforming compiler, several compilation limits have been given lower bounds. These are described in Table 1, page 18. In addition, all types have been given minimum ranges, which are provided to a program through a header file. Basically, *char* objects are a minimum of 8 bits, *short*s are 16 bits or more, *int*s are at least as big as *short*s, and *long*s are a minimum of 32 bits.

The Preprocessor

The preprocessor has always been a part of C but separate. Because of this, and its lack of certain features, the preprocessor has fostered more implementation differences than any other part of C except the library. Now the preprocessor is a part of the language, specified with the same rigor as any other part.

Simple *#define* directives are the same as before—you can still define an identifier to mean something else. What has changed is the range of possibilities. The biggest changes are "string-izing," token pasting, and recursive definitions.

Because it's often convenient to be able to use a parameter both inside and outside a string, a string-izing preprocessor operator has been defined. Consider the following example:

```
#define val(x) printf( "x = %d\n", x )
```

val(xyzzy) is replaced either by:

```
printf( "xyzzy = %d\n", xyzzy )
```

or by:

```
printf( "x = %d\n", xyzzy )
```

In the past, compilers could not be relied upon to replace the occurrence of a formal argument within a string literal with the actual argument. In fact, K & R specified that string literals were just that and they should not be scanned for replacement. The facility is useful, however.

So, # occurring before the name of a formal argument is now replaced by the actual argument, as in the following example:

```
#define val(x) printf( #x " = %d\n", x )
```

is always replaced by:

```
printf( "xyzzy" " = %d\n", xyzzy )
```

which is equivalent to:

```
printf( "xyzzy = %d\n", xyzzy )
```

This example also shows another new feature—adjacent string concatenation. The standard specifies that adjacent (in the token sense, disregarding white space) string literals shall be combined into a single string literal. This means that *"1" "2" "3"* is identical to *"123"*.

Another useful but nonstandard feature of some implementations is token pasting—the creation of a single token from two separate ones. Many C compilers provide this facility with the following mechanism:

```
#define x1 23
#define m(z)z/**/1
```

Here *m(x)* is replaced by *x1*, which in some compilers is replaced by *23*. The new, approved way to do the replacement all the way to *23* is:

```
#define x1 23
#define m(z)z##1
```

so that *m(x)* produces *23*.

In the past the single most reliable way to break a C compiler (and use up inordinate amounts of CPU time and memory space) was to write a sequence such as:

```
#define x x
x
```

This should no longer wreak havoc. The standard says that a macro, once expanded, turns itself off for the

(continued from page 17)

duration of rescanning. This has potential usefulness in statements such as the following:

```
#define sizeof   (int) sizeof
#define char     unsigned char
```

These fragments depend on another new feature of the draft—that of defining keywords.

Include files are related to both the preprocessor and library issues. The standard now specifies which include files must exist and their contents. The list of include files that must exist is provided in Table 2, below. Note that the files from this list are the only required files and should be used when referring to a standard function or macro. I cannot recommend strongly enough that you should include these files when a definition is needed.

Including these files has several advantages. First, a prototype is provided for each of the related functions, which aids in diagnosing improper library usage. (I'll discuss prototypes later in this article.) Second, some functions, such as *printf*, require a prototype before use because they take a variable number of arguments. Such functions cannot be called with maximum efficiency on some machines (including the 8086), and therefore the compiler is allowed to require you to specify when it must use suboptimal code. Third, many compilers already (and more will) define "internal" functions for important library functions.

Such functions as *memcpy* and *strcpy* can often be performed in-line (without a function call) with a great increase in speed and very little increase in code space. (MetaWare, for example, already uses in-line function expansion for some functions.) Fourth, many library functions do not modify one or more arguments. In such situations, the compiler need not worry about such functions disrupting an optimization because the library declarations in the header files provide the compiler with the needed information. Without this information, the compiler must assume that the function modifies anything it can.

Many of these advantages may not be present in compilers for some time, but rest assured that they will be eventually. Many vendors that build Pascal and C compilers could easily take advantage of the more optimal calling convention provided by Pascal. And many recently released compilers support type-checking features to aid library usage diagnosis.

Module-Scope Declarations

The next major section of your program, after the *#includes* and *#defines*, is usually module-level declarations and definitions, which is where variables manipulated by the source file are declared or defined. The standard's principle changes here concern types.

Characters still have implementation-defined signedness, although all types from *char* through *long* may have specified signedness via the *unsigned* and *signed* type modifiers. *Signed* is required only for *chars* and bit fields as all other types default to *signed.*

There are now three floating-point types—*float*, *double*, and *long double* (note that *long float* is no longer acceptable).

There is also a new type called *enum*, which allows programmers to associate named values (not unlike *#defined* values) to variables of *enum* type. No new functionality is created here, only an aid to documentation. The classic example is:

```
enum color { red, blue, green, white };
enum color pixel;
```

Here, *enum color* is a type (just as *struct tag* is) to which the values *red*, *blue*, *green*, and *white* apply, and *pixel* is an instance of an *enum color* variable. Certain features of *enum* are important to keep in mind. First, *red* can appear anywhere an integer constant can appear, even in expressions that do not involve the *enum color* type. As a result, only one *enum* can declare the name *red*. Further, no variables can be so named. A variable with *enum* type behaves identically to an integer in any expression. If the compiler determines that a particular set of *enum* values fits in a smaller type, it is free to allocate less space to any objects of that *enum* type. Last, you can specify values for any or all of the names in *enum*.

The New Void

The ANSI standard also gives C a *void* type, which is useful in three contexts: function return types, pointed to types, and prototypes.

Many existing C compilers implement *void*-type functions. These "functions" are really procedures that do things but don't return a value. Examples include *exit* and *longjmp* (which don't return at all) and *free*.

Void pointers are

15	"levels" of statements
6	nesting levels of conditional compilation directives
12	type modifiers per declarator
127	levels of parentheses
31	characters in 2 cases of significance in internal names
6	characters in 1 case of significance in external names
511	external names in one source file
127	local variables per block
1024	macros per source file at any time
31	parameters to either a function or macro
509	characters in a "logical" source line
509	characters per string constant
32767	bytes in a single object
8	nesting levels of include files
257	cases in a switch statement

Table 1: *Lower bounds of several compilation limits*

float.h
limits.h
stddef.h
assert.h
ctype.h
locale.h
math.h
setjmp.h
signal.h
stdarg.h
stdio.h
stdlib.h
string.h
time.h

Table 2: *Include files that must exist*

(continued from page 18)

pointers that do not point at anything in particular but that can point to anything. *Malloc* returns a *void* pointer, and *free* accepts one. The neat thing about *void* pointers is that they can be assigned to any pointer and can be assigned from any pointer. So all those *malloc*, *calloc*, and *realloc* calls no longer need to have casts in front of them. And now you finally have a way to store a pointer without having to invent some fake type for it.

The last use of *void* is in prototypes, which I'll discuss later in the section on function definitions.

Const and *volatile* are new keywords that are used in variable declarations and definitions. *Const* tells the compiler that the associated object cannot be modified (by the compiler's generated code). It serves mostly to aid the compiler in determining where objects can be placed and to help the compiler warn of an attempt to modify the object—although a clever compiler realizes that once the value has been fetched, it need never be fetched again. *Volatile* is a directive to the compiler that every programmed read and write of the object must take place as specified. A compiler cannot optimize out reads or writes to such objects. Usually, volatile objects are either I/O objects (such as a UART registers) or semaphores of some sort.

External objects have also changed in a couple of ways. First, each such object must be "defined" exactly once. A definition is a declaration outside the scope of any function that does not include the keyword *extern*. Unlike Unix, you cannot (portably) define an object more than once. Second, each such object must (unenforceably) be unique in the first six (yes, 6) characters without regard to case distinctions. Note that this restriction does not apply to any "internal" names.

Function Definitions

The use of prototypes is probably the most significant change to the language that the committee has made. Using them will improve documentation, help the compiler detect stupid mistakes, generate better code, enhance portability, and assure compatibility with future C standards. I alluded to many of these features when I discussed include files.

A prototype improves documentation because it declares the type (and optionally the name) of the arguments to a function. For example, the standard function *memcpy* could have the following prototype declaration:

```
void *memcpy( void *dest, const void *src, size_t n );
```

This declaration tells the compiler that *memcpy* is a function that returns a pointer to an unspecified object. It takes exactly three parameters—the first has the name *dest*, the second *src*, and the third *n*. The first two parameters are pointers to unspecified objects, and the last is an integral type large enough to hold the number of bytes in the machine. Finally, the second parameter is a pointer to an area in memory not modified by the function.

Not including a prototype for a function that takes a variable number of arguments could be fatal. The reason is this: C is almost unique in its ability to deal with functions that take a variable number of arguments. Pascal, Modula-2, Ada, and FORTRAN cannot deal with user-provided functions that take a variable number of arguments. Because of this, the callee (the function being called) can't use instructions designed to clean up the stack on function exit. In the 8086, for example, the *RET* instruction has an optional operand that specifies the number of bytes to be added to the stack pointer after the return address has been fetched. Instead of this, the function has to use a simple *RET*, and the calling function (say, *main*) has to clean up the stack when it gets control back, usually with an *ADD SP,n* instruction. Not only is this sequence slower but it is also larger because the *ADD* appears everywhere a call to *printf* occurs rather than once at the end of *printf*. Of course, this requirement does mean that if the caller and the callee disagree about the number of arguments, at least the stack doesn't get misaligned.

To let C use the faster return mechanism, ANSI says, in essence, that you must tell the compiler the names of all functions that take a variable number of arguments. The way you do this is by supplying a prototype. If the last argument in a prototype is . . . , then the compiler knows that the function takes a variable number of arguments and therefore that it must use the larger, slower return mechanism. It is important to note that the standard specifies that the compiler, upon encountering a call to a function that it does not have a prototype for, can assume that the function takes a fixed number of arguments of the supplied types (after promotion).

The header files supplied with an ANSI-conforming compiler include prototype information for each function in the library.

The last problem with prototypes is how you specify that a function takes no arguments because *void foo( );* is already legal. This is where the keyword *void* shows up again. To declare a function that takes no arguments, you place *void* between the parens.

Function Bodies

The only changes to the language that affect actual code are all enhancements except for one, which is a clarification.

Switch expressions can be any integral type, up to and including *unsigned long*. Floating-point expressions can be evaluated in any of the floating-point types. Structures (and unions) can be passed to and returned from functions or assigned. Arrays can have their addresses taken, and the resultant value is of type *pointer to array*, which is decidedly different from *pointer to first element of array*.

Two subtle changes are of great importance to 8086 programmers. First, *sizeof* does not return an *int* (it is *size_t*, defined in stddef.h), and *malloc* and related functions expect *size_t*-type quantities when dealing with numbers of objects (usually *chars*). The other change is that pointers to functions are different from pointers to data.

The only new operator is unary +. With it you can force the compiler to evaluate subexpressions by themselves instead of in the context of some larger expression. For example, *a*b/c* could be evaluated either by multi-

TEACH AN OLD DOS NEW TRICKS

Protected Mode and 32-bit Performance Today

Introducing OS/286™ and OS/386™, extensions to MS-DOS 3.x that enable full use of the 80286 and 80386. Now you get direct access to all available memory, not just an archaic 640K.

OS/286 and OS/386 propel your programs beyond the limitations of DOS, without forcing you to start all over.

Moving to protected mode is simple because OS/286 and OS/386 give you *the same interface as DOS*. The hardware is still under your direct control, many 16-bit compilers already generate code suitable for OS/286 and OS/386, and existing highly tuned, machine-specific subroutines running in real mode can be efficiently called from within protected mode. Since most of your own code won't need to be rewritten, your programming investment is preserved. And because OS/286 and OS/386 work *with* DOS 3.x, they don't affect other programs, device drivers, or TSRs.

In addition to the larger address space offered by protected mode, OS/386 adds 32-bit performance to systems like the Compaq™ 386 which, until now, have been shackled to 8086 emulation.

Dhrystone Benchmarks	Microsoft C 16-bit	High C OS/386 32-bit	Scale
IBM AT 6Mhz	793	na	1.0
Compaq 386-16Mhz	2,380	5,837	7.3
HummingBoard-16Mhz	2,777	6,718	8.5
HummingBoard-20Mhz	3,571	8,470	10.7
Vax 8600 (Unix 4.3 BSD,cc)		6,423	8.1
Sun 3/160 (Sun 4.2 3.0A,cc)		3,246	4.1

OS/386 can be customized to give <u>unmodified</u> DOS programs up to 900k on 386 systems, regardless of how many TSR's, networks, disk caches, etc., are installed.

OS/286™ and OS/386™ Features:

- Huge address space (4GB on the 80386)
- 32-bit performance (80386)
- No rewriting of device drivers
- Compact code (under 64k)
- Support for all existing DOS calls
- New INT-21 calls for manipulating segments, invoking real-mode routines and interrupt handlers, and addressing physical memory directly
- Full interrupt vector support
- Powerful debugging: concurrent DOS environment while debugging protected mode programs
- The ability to run non-Windows programs in a window

A.I. Architects gives you a complete development toolkit:

OS/286 or OS/386 kernel and linker, Symbolic debugger and command processor

Options include:

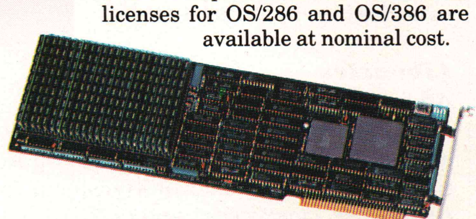
16-bit and 32-bit compilers

High C, Professional Pascal, or F77L FORTRAN

32-bit Assembler

386 HummingBoard™

The basic Developer's Kit is $495. 32-bit Compilers are $895. Run time licenses for OS/286 and OS/386 are available at nominal cost.



A.I. Architect's HummingBoard™ is a high performance 386 coprocessor for the PC-XT, AT and compatibles available with the 80387 and 2-24 Mbytes of RAM.

A.I. Architects, Inc.

One Kendall Square
Cambridge, Massachusetts 02139
(617) 577-8052

OS/286, OS/386 and HummingBoard are trademarks of A.I. Architects, Inc., Compaq Deskpro 386 is a trademark of Compaq Computer Corp., High C and Professional Pascal are trademarks of Metaware, Inc., F77L FORTRAN is a trademark of Lahey Computer Systems, Inc., Microsoft and MS-DOS are trademarks of Microsoft Corp., VAX 8600 is a trademark of Digital Equipment Corp., Sun 3/160 is a trademark of Sun Microsystems, Inc., Unix is a trademark of AT&T.

CIRCLE 265 ON READER SERVICE CARD

(continued from page 20)

plying *a* and *b* then dividing by *c*, or by dividing *b* by *c* and then multiplying by *a*. Now you can specify the order, if you think it is important. The expression *a * +(b/c)* (note the unary plus) forces the compiler to do the divide, whereas *+(a * b)/c* forces the compiler to do the multiply. This construct does not force the compiler to do either one first (although it is hard to imagine how these examples could be done otherwise), it merely forces the compiler to fully evaluate the subexpression by itself without combining it with any other components of the expression.

The last item I'll mention here is unsigned vs. value-preserving conversions. The following code has two possible answers, depending on which of the two conversion rules you apply:

```
unsigned char uc;
int i, x;

uc = 2;
i = -1;
x = ((uc * i) / 2);
```

The only place in which this is a problem is when an unsigned quantity is expanded in size in the course of one subexpression and this subexpression is expanded yet again; shifted right; or is an operand of /, %, <, <=, >, or >= as part of a larger expression.

In the unsigned-preserving case, *(uc * i)* produces an unsigned *int* (value 0xfffe), which when divided by 2 yields 0x7fff and gives (signed) *x* the value 32,767. In the value-preserving case, *(uc * i)* produces a signed *int* (if all possible unsigned *chars* are representable in a signed *int*) with value -2 (hex 0xfffe), which when divided by 2 yields -1 (hex 0xffff). This example shows a case in which value preserving is useful, but I could show other examples in which unsigned preserving is better behaved. All this is irrelevant if you use casts. In my example, the placement of an *int* cast before either the *uc* or the *(uc * i)* would have caused the same result to be produced in either case. Most code works equally well in either environment, so don't lose any sleep about this issue. In fact, in all the source code for Unix, I can't think of a single instance in which this matters.

Libraries

An important change to the C language is that a library is now specified. All hosted implementations must provide all the functions specified. Table 3, right, lists all the functions to be included in prototype form. Note that the *open, read, write, close, creat,* and *unlink* functions are all missing from this set. These were deemed the domain of the operating system and somewhat redundant, given *fopen* and so on.

Library function names and macros are reserved. The reason is so that function a in the library can reliably call function b without worrying that the programmer has replaced b with a function of his or her own. It also allows the compiler to recognize these functions and to generate special code for them if it wants to.

The name-space issue I mentioned under "General Items" will create some problems in the near term as compiler vendors try to decide what to do about the now "nonstandard" functions in their libraries that have names that do not belong to the implementation. Two possible solutions exist. The first is to deliver these functions as an add-on library, preserving their names in this library separate from the standard library. The other is to change the names of these functions in the standard library to have leading underscores and then provide header files that users can include that define each of the

assert.h	void assert(int expression)
ctype.h	int isalnum(int c)
	int isalpha(int c)
	int iscntrl(int c)
	int isdigit(int c)
	int isgraph(int c)
	int islower(int c)
	int isprint(int c)
	int ispunct(int c)
	int isspace(int c)
	int isupper(int c)
	int isxdigit(int c)
	int tolower(int c)
	int toupper(int c)
locale.h	char *setlocale(int category, char *locale)
math.h	double acos(double x)
	double asin(double x)
	double atan(double x)
	double atan2(double y, double x)
	double cos(double x)
	double sin(double x)
	double tan(double x)
	double cosh(double x)
	double sinh(double x)
	double tanh(double x)
	double exp(double x)
	double frexp(double value, int *exp)
	double ldexp(double x, int exp)
	double log(double x)
	double log10(double x)
	double modf(double value, double *iptr)
	double pow(double x, double y)
	double sqrt(double x)
	double ceil(double x)
	double fabs(double x)
	double floor(double x)
	double fmod(double x, double y)
setjmp.h	int setjmp(jmp_buf env)
	void longjmp(jmp_buf env, int val)
signal.h	void (*signal(int sig, void (*func)(int)))(int)
	int raise(int sig)
stdarg.h	void va_start(va_list ap, parmN)
	type va_arg(va_list ap, type)
	void va_end(va_list ap)
stdio.h	int remove(const char *filename)
	int rename(const char *old, const char *new)
	FILE *tmpfile(void)
	char *tmpnam(char *s)
	int fclose(FILE *stream)
	int fflush(FILE *stream)

Table 3: *All functions that must be included*

old function names in terms of the new names. I prefer the latter approach as it gives me the ability to edit the "old" names I wish to be visible without editing the library or my source files and I can also use the functions (nonportably) by using their _ names. It also lets implementors continue to rely on the existence of these functions by using the _ versions.

Conclusion

ANSI C is coming, and it is good. Unlike many (dare I say all) previous language standards, this effort looks as though it will genuinely help portability of C programs without harming most existing programs. I believe that you can write important programs that will run reliably on any computer that supports an ANSI C environment without changing even a single line of code.

One last warning: there can be no truly ANSI-conforming compiler until the standard is adopted. Any compiler vendor claiming conformance prior to that isn't telling the whole truth. Further, unless such compilers address the name-space issue, they never will be conforming.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 1.**

```
          FILE *fopen( const char *filename, const char *mode )
          FILE *freopen( const char *filename, const char *mode,
                                             FILE *stream )
          void setbuf( FILE *stream, char *buf )
          int setvbuf( FILE *stream, char *buf, int mode, size_t size )
          int fprintf( FILE *stream, const char *format, ... )
          int fscanf( FILE *stream, const char *format, ... )
          int ptintf( const char *format, ... )
          int scanf( const char *format, ... )
          int sprintf( char *s, const char *format, ... )
          int sscanf( const char *s, const char *format, ... )
          int vfprintf( FILE *stream, const char *format, va_list arg )
          int vprintf( const char *format, va_list arg )
          int vsprintf( char *s, const char *format, va_list arg )
          int fgetc( FILE *stream )
          char *fgets( char *s, int n, FILE *stream )
          int fputc( int c, FILE *stream )
          int fputs( const char *s, FILE *stream )
          int getc( FILE *stream )
          int getchar( void )
          char *gets( char *s )
          int putc( int c, FILE *stream )
          int putchar( int c )
          int puts( char *s )
          int ungetc( int c, FILE *stream )
          size_t fread( void *ptr, size_t size, size_tnmemb, FILE
                                             *stream )
          size_t fwrite( const void *ptr, size_t size, size_t nmemb,
                                             FILE *stream )
          int fgetpos( FILE *stream, fpos_t *pos )
          int fseek( FILE *stream, long int offset, int whence )
          int fsetpos( FILE *stream, const fpos_t *pos )
          long int ftell( FILE *stream )
          void rewind( FILE *stream )
          void clearerr( FILE *stream )
          int feof( FILE *stream )
          int ferror( FILE *stream )
          void perror( const char *s )
stdlib.h  double atof( const char *nptr )
          int atoi( const char *nptr )
          long int atol( const char *nptr )
          double strtod( const char *nptr, char **endptr )
          long int strtol( const char *nptr, char **endptr, int base )
          unsigned long int strtoul( const char *nptr, char**endptr, int
                                             base )
          int rand( void )
          void srand( unsigned int seed )
          void *calloc( size_t nmemb, size_t size )
          void free( void *ptr )
          void *malloc( size_t size )
          void *realloc( void *ptr, size_t size )
          void abort( void )
          int atexit( void ( *func )( void ))
          void exit( int status )
          char *getenv( const char *name )
          int system( const char *string )
          void *bsearch( const void *key, const void*base, size_t
                 nmemb, size_t size, int (*compar )( const void *,
                                             const void *))
          void qsort( void *base, size_t nmemb, size_t size, int
          ( *compar )( const void *,
          const void *))
          int abs( int j )
          div_t div( int nymer, int denom )
          long int labs( long int j )
          ldiv_t ldiv( long int numer, long int denom )
string.h  void *memcpy( void *s1, const void *s2, sizen )
          void *memmove( void *s1, const void *s2, size_t n )
          char *strcpy( char *s1, const char *s2 )
          char *strncpy( char *s1, const char *s2, size_t n )
          char *strcat( char *s1, const char *s2 )
          char *strncat( char *s1, const char *s2, size_t n )
          int memcmp( const void *s1, const void *s2, size_t n )
          int strcmp( const char *s1, const char *s2 )
          int strncmp( const char *s1, const char *s2, size_t n )
          size_t strcoll( char *to, size_t maxsize, const char *from )
          void *memchr( const void *s, int c, size_t n )
          char *strchr( const char *s, int c )
          size_t strcspn( const char *s1, const char *s2 )
          char *strpbrk( const char *s1, const char *s2 )
          char *strrchr( const char *s, int c )
          size_t strspn( const char *s1, const char *s2 )
          char *strstr( const char *s1, const char *s2 )
          char *strtok( char *s1, const char *s2 )
          void *memset( void *s, int c, size_t n )
          char *strerror( int errnum )
          sizet strlen( const char *s )
time.h    clock_t clock( void )
          double difftime( time_t time1, time_t time0 )
          time_t mktime( struct tm *timeptr )
          time_t time( time_t *timer )
          char *asctime( const struct tm *timeptr )
          char *ctime( const time_t *timer )
          struct tm *gmtime( const time_t *timer )
          struct tm *localtime( const time_t *timer )
          size_t strftime( char *s, size_t maxsize, const char *format,
          const struct tm *timeptr )
```

Backtracking

by Charles F. Bowman

Use this approach when the existence of a solution isn't guaranteed.

Most of the time, programming is a straightforward matter. You analyze the problem, select the appropriate data structures and algorithm, and—after a certain amount of work—you've finished. Granted, the first solution might not be as fast as you'd like, or as elegant, but at least you have the advantage of knowing the problem is solvable. But what about those occasions when the path to a solution isn't so clear? This article is about a programming method—called backtracking—that is commonly used in AI programming. In contrast to normal methods, in which you program all the steps required to attain your goal, you can use this approach when even the existence of a solution can't be guaranteed.

Backtracking belongs to a general class of programming methods termed *nondeterministic programming* (NDP). In NDP you don't code the solution to your problem—you program a method that will lead to a solution. The program literally makes guesses until it either finds a solution or exhausts the available alternatives. Moreover, there can be more than one solution for a given problem. This method has obvious benefits in AI theory and expert systems development.

Backtracking is a programming method in which you proceed along a given "path" searching for a solution. At each fork in the road, you make a guess as to which path you should follow to continue your search. If this choice should prove unsuccessful—that is, if you encounter a dead end—you back up and try a different path. The execution continues in this manner until you either reach a solution or exhaust all the possible choices. The latter condition signifies that no solution exists, and the program should exit with an indicative status. If you think that this sounds similar to a depth-first traversal, you are correct. The only significant difference is that with backtracking the decision tree is implicit rather than explicit.

Charles F. Bowman, 24 Jacques Ave., Staten Island, NY 10306. Charles is a consultant and is currently writing a textbook on data structures. He holds an M.S. degree from New York University.

```
 1: bktkfind( node )
 2: begin
 3: if( node = SUCCESS )
 4: then
 5:      return( I_FOUND_IT )
 6: endif
 7: for( each_choice_at_this_
                         node )
 8: do
 9:      ret_stat = bktkfind
                   ( child_node )
10:      if( ret_stat = SUCCESS )
11:      then
12:           return( ret_stat )
13:      endif
14: done
15: return( FAIL )
16: endproc
```

Example 1: *Pseudocode for a chronological backtracking function*

There are two types of backtracking: chronological backtracking (CBT) and dependency-directed backtracking (DDB).

Chronological Backtracking

CBT is effectively an exhaustive search, similar to that discussed earlier. Each solution path is attempted, in what is tantamount to a random order, until one of two outcomes is determined.

Consider the pseudocode in Example 1, below, for example. If at any time a solution is found (lines 3 – 6, 9 – 13), the function returns a value indicating success. If not, it must try an alternate choice (lines 7 – 14). If all the alternatives have been exhausted (line 7), a value indicating failure is returned, forcing the function to back up to a previous path (line 15) before continuing the search.

There are two important points to consider here. First, whenever you perform a backup, you must restore the previous environment before trying the next path. Obviously, this can become very expensive. Second, backtracking is typically implemented using a recursive procedure, which yields an algorithm that is exponential in order of execution magnitude (also costly). The following paragraphs discuss methods of improving the basic algorithm.

Dependency-Directed Backtracking

Dependency-directed backtracking (DDB) works essentially as described earlier but tries to eliminate some unnecessary searching in two ways.

First, as the name implies, you can backtrack to choices that are dependent on the dead end. That is, you back up until you reach a point at

which a dependency was created and continue searching from there. As an example, consider a case in which you are searching for a solution that requires four conditions (A, B, C, and D) to be satisfied. Further, assume you reach a state in your processing at which conditions A and B are satisfied but C and D are not. In lieu of just automatically backtracking to the previous fork, you continue on to a point at which A and B are still true and resume the search from there. You can skip all the intervening paths.

The second way to eliminate unnecessary searching is called pruning. If you reach a point in the search at which it becomes obvious that any further effort on a given path is fruitless, you can eliminate the remainder of the subtree from that point onward (that is, force a backtrack to occur). Pruning is a straightforward approach and is often implemented in game-playing simulations. You could, for example, write a chess program that could determine its next move by assigning a quantum value to each board position it examines. At any given point, it would select the move that yields the most advantageous (highest) value. If the algorithm were to traverse a path representing the loss of a player's queen, it could elect to eliminate any further searching along that trail.

For the sake of completeness, I should also mention a third method of improving a backtracking procedure: explicitly managing a stack. Recursive procedures are costly because of the considerable amount of overhead required for each successive call. Your program must save registers, store a return address, allocate local storage, and so on. Most of this information is not directly related to the problem at hand and, therefore, having to save and restore it only wastes CPU cycles. You can save time and space (at the expense of programmer effort—there really is no such thing as a free lunch!) if you code the stack explicitly. You can accomplish this easily by transforming the algorithm from its recursive form to an iterative one and maintaining the to-do list in an application-controlled data structure. (Note that recursion is a really just a form of iteration.)

SETL

My first encounter with backtracking occurred when I attended graduate classes at New York University. The students and faculty had developed a language called SETL (Set Language), which featured a pair of built-in primitives (*OK/FAIL*) that supported backtracking. As an example, consider the classic eight-queens puzzle. The challenge is to place all eight queens on a chess board such that no two queens are attacking each other. Example 2, below, presents an SETL solution to the problem.

Some of the constructs might appear strange, but what's important to note is how the two primitives, *OK* and *FAIL*, can free the programmer from dealing with some of the low-level details that would otherwise be required. Line 3 of the example initializes a language-controlled stack so that each time the statement *OK* is executed, a snapshot of the execution state is saved. (Actually, each variable that you want saved must be identified at declaration time.) Lines 4, 5, and 6 compute all the currently unattacked (that is, available) positions and stores them in the variable *unattacked*. If none exist for a given board configuration (line 7), the function executes the *FAIL* statement on line 9 and backtracks; if *unattacked* is nonempty, one position is selected randomly (line 11) and stored in *posn*. When the *forall* loop terminates, the board positions are printed.

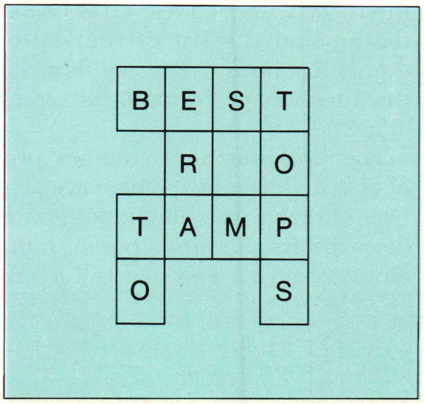


Figure 1: *Sample puzzle*

```
 1: posn := [];
 2: (forall j in [ 1..8 ])
 3:   { j : j in [ 1..8 ] | OK }
 4:   unattacked := { 1..8 } -
             { posn(k) + (j-k)*slope
 5:             : k in [ 1..j-1 ],
 6:             slope in [ -1..+1 ] }
 7:   if( unattacked = {} )
 8:   then
 9:     FAIL;
10:   else
11:     posn(j) := ord unattacked;
12:   endif;
13: end forall;
14: print( posn );
```

Example 2: *Eight-queens problem*

An Acrostic Example

As is customary in *DDJ* articles, I've included the source code for a demonstration program (see Listing One, page 50). This program is, however, slightly less serious than those you usually find in *DDJ*. The program, called kross, solves acrostic puzzles using a backtracking algorithm. You're probably familiar with this type of puzzle—examples can be found in just about every newsstand puzzle magazine. If you're not, an acrostic puzzle is simply a crossword puzzle without the clues: you are supplied with the words and the diagram and, through trial and error, you must enter all the words into their appropriate slots (see Figure 1, below).

The program requires one argument—the name of the file containing a description of the puzzle (yes, you have to type the puzzle in!). The file is divided into two sections. The first is a description of the puzzle diagram: a series of lines—one for each row—containing either blanks or minus signs (this can be modified by the user, if desired). These characters represent the black boxes and the actual character locations, respectively. The second section is just the list of words, one per line, that the program will use to solve the puzzle; they can be entered in any order.

A couple of notes: all puzzle-description lines must be of equal length (the program checks for this). Also, take the time to ensure that all the words are spelled correctly. The program, as you would expect, is unforgiving in this regard.

Example 3, page 26, contains a sample input file for the puzzle of Figure 1, and Example 4, page 26, contains the output generated by the program.

BACKTRACKING

(continued from page 25)

The overall operation of the program is straightforward: read the puzzle and word list into internal data structures; search for a solution; and if there is a solution, print it. The actual backtracking logic is in the function *solve( )*. This is a recursive procedure that:

- Chooses and determines the size of the next puzzle slot to fill (horizontal or vertical). This processing is performed by the function *next( )* and is by necessity a rather messy bit of code.
- Selects at random (that is, sequentially) an appropriately sized word from the available list. The function *itfits( )* is called to ensure that a given word fits into the slot (in typical crossword puzzle fashion).
- If the word fits, enters it into the puzzle. At this point, with the aid of the function *enter( )*, a snapshot of the current state (puzzle) is saved.
- Recursively calls itself to continue toward a solution.
- If at any point a solution is found (that is, there are no more slots to fill), returns the value *SOLVED*.
- If a recursive call fails to find a solution, the puzzle is restored to its previous state (*restore( )*); the word is returned to the free list; the next available word is selected; and if none remain, the function returns the value *FAIL* to its caller.

Let's trace the execution of the function *solve( )* as it begins to solve a sample puzzle. The line numbers that follow refer to Example 5, below. Also, the random selection of the words is in the order in which they appear in Figure 1.

First, a four-letter word is needed for the 1-across position. The function randomly selects *best* (line 14), marks it as *USED* (line 16), and inserts it into the puzzle (line 17). A recursive call is then made to continue the processing (line 19). Next, for the 2-down position, a three-letter word is needed and *era* is similarly inserted into the puzzle.

The function now moves to fill the 4-down position. It selects the next available four-letter word, *tamp* (line 13); checks to see that it fits (line 14); and inserts it also into the puzzle (line 17).

The next slot to fill is 5 across, and the program, as usual, selects the next available four-letter word—in this case *tops*. This time, however, the *itfits( )* (line 15) test fails. Recognizing that the list of four-letter words has been exhausted (line 13), the function performs a backtrack (line 27).

The program now resumes processing at the point at which it, again, needs to solve the 4-down position. It discards its current choice, *tamp* (lines 22 and 23), and selects the next available word, *tops* (line 14). From this point on, the program solves the puzzle without any additional difficulties.

Summary

Backtracking can be an extremely powerful tool for programmers, although it is always an expensive solution. Nonetheless, it can be a tool that enables you to solve problems that would be otherwise unsolvable.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063 or call (415) 366-3600 ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro). If you would rather not have to retype the entire program and/or you would like some sample puzzles to work with, send me a check for $6 and I will mail you an MS-DOS floppy (360K format) containing the program, source code, and several sample puzzles.

DDJ

(Listing begins on page 52.)


```
@puzzle
----
$-$-     (`$' = Blank)
----
-$$-
@words
best
tamp
tops
era
to
```

Example 3: *Sample input*

```
best
$r$o    (`$' = Blank)
tamp
o$$s
```

Example 4: *Sample output*

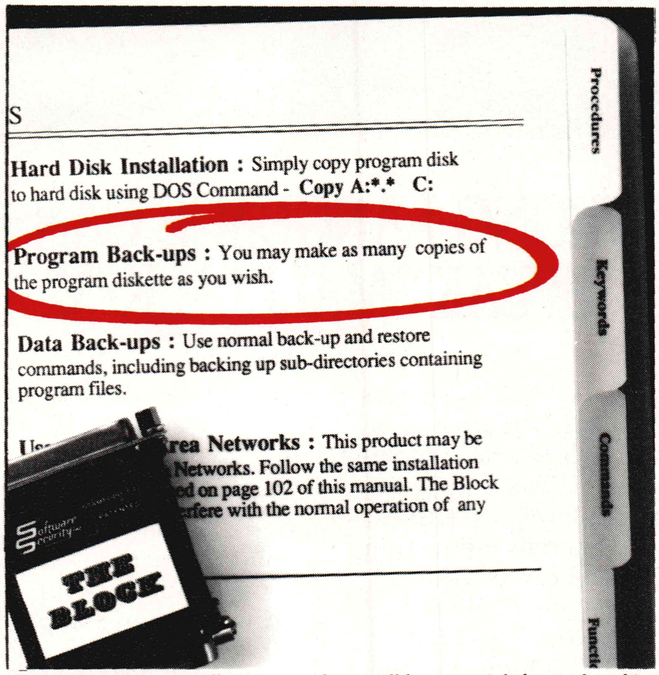
```
 1: solve( length, width )
 2: int  length, width;
 3: {
 4:   int  l, w, i, len, tmp, type;
 5:   char old[ WORDLEN - MINWORD + 1 ];
 6:
 7:   w = width;
 8:   l = length;
 9:   len = next( &l, &w, &type );
10:   if( len == 0 )
11:     return( SOLVED );
12:
13:   for(i = 0;i<MAXWORD&&WORD(len,i)[0]!=NIL;i++){
14:     if( FLAG(len, i) == FREE
15:     && itfits(l, w, WORD(len, i), type) ){
16:       FLAG(len, i) = USED;
17:       enter(old, l, w, WORD(len,i), type);
18:       prev = type;
19:       tmp = solve( l, w );
20:       if( tmp == SOLVED )
21:         return( SOLVED );
22:       restore( old, l, w, type );
23:       FLAG(len, i) = FREE;
24:     }
25:   }
26:
27:   return( 0 );
28: }
```

Example 5: *The function* solve( )

"How to protect your software by letting people copy it."

By Dick Erett, President of Software Security

Inventor and entrepreneur, Dick Erett, explains his company's view on the protection of intellectual property.



Soon all software installation procedures will be as straightforward as this. The only difference will be whether you include the option to steal your product or not.

"A crucial point that even sophisticated software development companies and the trade press seem to be missing or ignoring is this:

Software protection must be understood to be a distinctively different concept from that commonly referred to as copy protection.

Fundamentally, software protection involves devising a method that prevents unauthorized use of a program, without restricting a legitimate user from making any number of additional copies or preventing program operation via hard disk or LANs.

Logic dictates that magnetic media can no more protect itself from misuse than a padlock can lock itself.

Software protection must reside outside the actual storage media. The technique can then be made as tamper proof as deemed necessary. If one is clever enough, patent law can be brought to bear on the method.

Software protection is at a crossroads and the choices are clear. You can give product away to a segment of the market, or take a stand against the theft of your intellectual property.

"...giving your software away is fine..."

We strongly believe that giving your software away is fine, if you make the decision to do so. However, if the public's sense of ethics is determining company policy, then you are no longer in control.

We have patented a device that protects your software while allowing unlimited archival copies and uninhibited use of hard disks and LANs. The name of this product is The BLOCK™.

The BLOCK is the only patented method we know of to protect your investment. It answers all the complaints of reasonable people concerning software protection.

In reality, the only people who could object are those who would like the option of stealing your company's product.

"...eliminating the rationale for copy-busting..."

Since The BLOCK allows a user to make unlimited archival copies the rationale for copy-busting programs is eliminated.

The BLOCK is fully protected by federal patent law rather than the less effective copyright statutes. The law clearly prohibits the production of work-alike devices to replace The BLOCK.

The BLOCK attaches to any communications port of virtually any microcomputer. It comes with a unique customer product number programmed into the circuit.

The BLOCK is transparent to any device attached to the port. Once it is in place users are essentially unaware of its presence. The BLOCK may be daisy-chained to provide security for more than one software package.

Each software developer devises their own procedure for accessing The BLOCK to confirm a legitimate user. If it is not present, then the program can take appropriate action.

"...possibilities... limited only by your imagination..."

The elegance of The BLOCK lies in its simplicity. Once you understand the principle of The BLOCK, hundreds of possibilities will manifest themselves, limited only by your imagination.

Your efforts, investments and intellectual property belong to you, and you have an obligation to protect them. Let us help you safeguard what's rightfully yours. Call today for our brochure, or a demo unit."

Software Security inc.
870 High Ridge Road Stamford, Connecticut 06905
203 329 8870

USING THE WRONG LANGUAGE CAN BE MURDER. SPEAK SMALLTALK/V.

Let's talk languages. Programming languages like Turbo Pascal, C or Basic can be killers. To many, they're foreign, complex, and generally intimidating. Mistakes can be deadly.

With Smalltalk/V, you have an elegantly simple solution that puts the power and majesty of a major AI programming language on your PC or compatible. It makes no difference if you're an experienced programmer or just getting started. Smalltalk/V gives you an easy-to-use and flexible programming tool.

This is the same language used by leading software companies for their new product development. There are sound reasons for this. Smalltalk/V offers a totally integrated programming environment using the premier object-oriented language. You use natural language rather than complex programming codes. It puts Macintosh-type graphic features on a PC including overlapping windows, bit-mapping, pop-up menus, and a mouse interface. More than mere window dressing, Smalltalk/V delivers fully interactive windows that are easy to build and quick to modify.

But don't just take our word on it. Hear what the experts have to say:

"This is the real thing folks. A super Smalltalk like this turns your PC into a hot workstation. It's fantastic... Highly recommended."

John Dvorak
Contributing Editor
PC Magazine

"The tutorial provides the best introduction to Smalltalk available."

Dr. Andrew Bernat
AI Expert Magazine

"Smalltalk/V is the highest performance object-oriented programming system available for PCs."

Dr. Piero Scaruffi
Chief Scientist
Olivetti Artificial
Intelligence Center

Today, thousands of professionals, scientists and engineers are using Smalltalk/V to solve both simple and expert problems. Giving them a new dimension in computer applications for their PC.

Put new life into your PC by calling toll free 1-800-922-8255 and ordering Smalltalk/V today. Smalltalk/V by Digitalk, Inc., 9841 Airport Blvd., Los Angeles, CA 90045.
(213) 645-1082.

$99.95

Smalltalk/V comes with 10 starter applications including Prolog and each Application Pack adds several more. All source code is included. Supports 640 x 480 color graphics with color extension pack.

Smalltalk/V requires DOS and 512K RAM on IBM PC/AT/PS or compatibles and a CGA, EGA, Toshiba T3100, Hercules, or AT&T 6300 graphic controller. A Microsoft or compatible mouse is recommended. Not copy protected.

Turbo Pascal is a trademark of Borland Internation. IBM, IBM PC/AT/PS are trademarks of International Business Machines Corporation. Macintosh is a trademark of Apple Computer, Inc.

CIRCLE 127 ON READER SERVICE CARD

TO ORDER CALL 1-800-922-8255 TODAY.

60-DAY MONEY-BACK GUARANTEE*

Send check, money order, or credit card information to: Digitalk, Inc., 9841 Airport Boulevard, Los Angeles, CA 90045.

Credit Card ☐ VISA ☐ Mastercard

Card number: ____________________

Expiration date: ____________________

Name: ____________________

Street Address: ____________________

City/State/Zip: ____________________

Smalltalk/V	$99.95
RS-232 Communications Application Pack	$49.95
EGA/VGA Color Extension Pack	$49.95
"Goodies" Application Pack	$49.95
SPECIAL OFFER: Smalltalk/V and all 3 packs only	$199.95
Shipping and handling (Outside North America	$5.00 ($15.00)
California residents add applicable sales tax	
TOTAL	$______

*Unconditional 60-day money-back guarantee. Simply return to Digitalk, Inc. and your refund will be immediately forwarded to you.

Smalltalk/V

digitalk inc.

What's the DIFF?

A text file differencer and change-bar tool in C

by Don Krantz

File comparison programs have been around for a long time, so you might reasonably ask, "Do we really need another?" If your work involves only small changes in a program's source code, the answer is probably "No"—a generic file comparison utility is just fine. In contrast, for those who write and update the documentation for that program, it can be an entirely different story.

This article got its start when we were faced with issuing revision documents for a large software project at work. The project had literally hundreds of associated documents (performance specs, interface specs, design specs, detailed design specs, design documents, development plans, and so on), and our customer requested that we mark the documents with change bars. (Change bars are vertical lines, usually in the right-hand margins, that mark the sections of text that have changed since the last release.) The request made sense: change bars save readers from having to make a laborious line-by-line comparison of the documentation looking for changed text. Still, it meant someone would have to make all those line-by-line comparisons to mark the text. Not really anxious to mark thousands of pages of documentation manually, I started looking for a good tool to change-bar the text automatically, or at least to locate the changes for me.

Don Krantz, 2845 42nd Ave. S, Minneapolis, MN 55406. Don is a principal computer applications engineer at Honeywell's Defense Systems Division. He is a coauthor of Ada: A Programmer's Guide with Microcomputer Examples *and principal author of* 68000 Assembly Language Programming, *both published by Addison-Wesley.*

I started my search by trying the DOS *COMP* command. It fired up, looked at the two versions of a file I gave it, and reported "FILE SIZES ARE DIFFERENT." With that, it exited, leaving me groping around my work area for the hammer I use on particularly annoying software.

The next place I looked was the BeeB (the TCOG bulletin board), reasoning that any system with 42 different directory programs was bound to have at least one file comparsion program that would fit my needs. I located a promising file, downloaded it, and gave it a try. It did a little better than *COMP* did, telling me that the files differed at line 64, before it too exited—still not quite good enough.

So I sat down and made a list of the things a suitable document comparison program should be capable of:

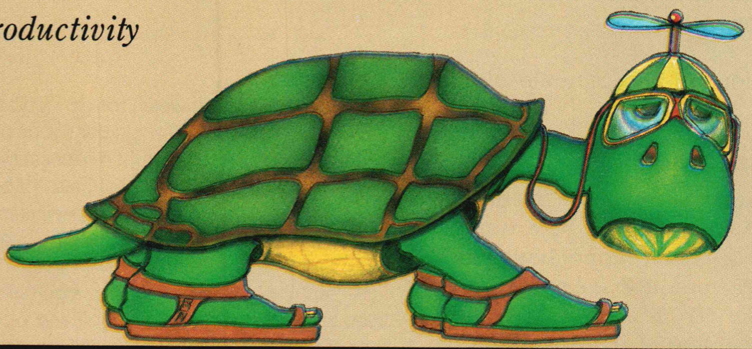
1. The program should know about the concept of pages and be able to recognize different sizes of pages by line count and form feeds.
2. It should be smart enough not to mark the header and footer lines just because the run date has changed.
3. It should be able to cope with repagination because of insertions and deletions.
4. It should be able to stay locked on identical text that is paginated differently, even with intervening headers and footers.
5. It should be able to deal intelligently with differing amounts of white space between paragraphs because of conditional paging.
6. It would be nice if you could turn case sensitivity on and off so that you could slide by corrections to trivial capitalization typos without drawing everyone's attention to them.
7. It would also be good to be able to ignore the table of contents because nobody expects to find change bars there.
8. The program should be capable of inserting change bars (or other marks) directly into an output file, without any manual massaging.
9. A change summary listing would also be desirable.

After looking at this long list of features, and at the software available, I decided that I'd have to write the program myself. The rest of this article, as you must have guessed, describes the program I came up with—DIFF.

Listing One, page 66 contains the program listing for DIFF. The source code provided has been tested on several large and small programs and documents under both MS-DOS and VAX/VMS, Version 4.3. It compiles without change under both Microsoft C, Version 4.00, and VAX-11 C because the VAX C compiler defines the macro *VAX11C* automatically, and I use this to key the differences between the two environments.

NO MATTER WHAT LANGUAGE OR DATA BASE PRODUCT YOU ARE USING — IF IT ISN'T CLARION, YOUR APPLICATIONS ARE TAKING TOO MUCH TIME TO WRITE.

CLARION:
- *saves development time*
- *increases programmer productivity*
- *lowers development costs*
- *yields better, richer applications for single users or a network of users*

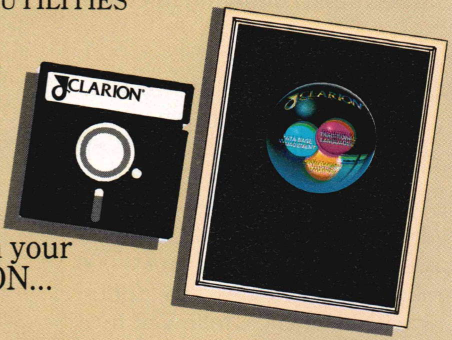


TRADITIONAL APPLICATION DEVELOPMENT TIME

CLARION TIME

THE CLARION ADVANTAGE

- POWERFUL MODERN LANGUAGE
- ADVANCED DATA BASE MANAGEMENT
- SCREEN and REPORT GENERATORS
- FAST COMPILE and TEST
- INTEGRATED FAMILY of UTILITIES



IF... your commercial microcomputer applications are written in Assembler, BASIC, C, COBOL, Pascal or any of the data base languages...

AND... you have simply run out of time... or programmers... or money...

THEN... you owe it to yourself and to those who depend on your professional skills — to make the easy move to CLARION...

ELSE... you'll miss out on the CLARION advantage.

CLARION is priced at $395 plus shipping. It runs on any IBM PC, XT, AT or true compatible with 320KB of memory and a hard disk drive.

To order CLARION or to get our free 16-page brochure and the Sample Program diskette, simply call TOLL FREE...

1-800/354-5444

CLARION®
from BARRINGTON SYSTEMS, INC.

150 EAST SAMPLE ROAD POMPANO BEACH, FLORIDA 33064-3597 305/785-4555

CIRCLE 115 ON READER SERVICE CARD

WHAT'S THE DIFF?
(continued from page 30)

Commands and Options

In its simplest invocation, DIFF takes two files as arguments. The first file is considered the "baseline," or original version of a document, and the second file is the "revision," or new version of the document (I am using *document* for both program source files and formatted documents).

DIFF's command line takes the following form:

DIFF [option option}] newfile oldfile [barfile]

where *newfile* is the name of the new release version of the document or source code. It must be a unique file name (that is, it cannot have wildcard characters), and it can include a path name. *Oldfile* is the name of the baseline version of the document or source code. It is subject to the same rules as *newfile* is. *Barfile* is optional, and if supplied, it indicates that you want an output file with change bars. The output file is a copy of *newfile* with change bars on new or modified lines. If *barfile* is not specified, no change-bar output will be created, but you'll still get the change summary output.

Some example command lines are:

```
DIFF FILE1.DOC FILE1.BAK
DIFF /BLANKS /LOOKAHEAD=20 FILE1.DOC FILE1.BAK
DIFF /BLANKS FILE1.DOC FILE1.BAK FILE1.PRN
```

Options take the form of VAX/VMS options, which are similar to MS-DOS command-line switches. An option begins with a slash (/), followed by a series of characters. If the option requires a numeric parameter (for example, the number of lines in a page), the series of characters is followed by an equal sign (=) and a number, without spaces.

Spaces between options are optional, and placement of the options on the command line is optional. Uppercase/lowercase in an option is not significant. Typical options look like this:

```
/BLANKS
/LOOKAHEAD=200
```

In addition, options can be abbreviated, as long as enough characters are used to make the option name unique. For instance, the previous examples could be abbreviated in the following ways:

```
/BL /B /BLA /BLAN . . .
/LO=200 /LOOK=200 . . .
```

Unrecognized options cause the program to print an error message and then halt. The entire command line is parsed before the program exits if errors occur.

DIFF options along with the syntax and default value of each option are listed in Table 1, below. DIFF error messages along with the cause of the error are listed in Table 2, page 34.

Output

DIFF has two forms of output—one standard and one optional. The

BAR_COL
Selects the column in which the change bar will be placed. The default is column 78. If column 0 is selected, the change bar will be placed at the left edge of the document or source code, moving text to the right if necessary. If a space or a tab is the first character on the line, the alignment of the text should not be affected. If a nonzero column is selected, the change bar will be placed in the specified column if possible. If text extends over the specified column, the change bar will be moved as far to the right as is necessary not to overwrite text.

A "feature" of the change-bar algorithm is that it can recognize and account for underlined text, at least the way Runoff and WordStar underline for "generic" printers, but not tabs. Tabs are counted as one column each. This produces amusing results on C source code.

For best results, on document files choose a column to the right of your text. On program source code with embedded tabs, choose column 0. This is admittedly an area in which the program could be improved. I needed the underline capability, and if the tab expansion is added, the code in *change—bar( )* gets uglier than it already is.

Example: *BAR_COL=0.*

/TOP_SKIP
Used for processing formatted documents. Its primary reason for being is to allow you to skip over the header line(s) in a document. Header lines confuse DIFF if they are left in place because DIFF doesn't know from chopped liver about headers unless you tell it, and if the pagination changes between the old and the new files, DIFF will cheerfully change-bar every header. Be sure to account for blank lines at the top of the page that precede the header line(s). The default value of */TOP_SKIP* is 0.

Example: */TOP_SKIP=3.*

/BOT_SKIP
Similar in use and purpose to the */TOP_SKIP* option. It specifies how many lines at the bottom of the page should be skipped. Count lines up from the bottom, not down from the top. The default value of */BOT_SKIP* is 0.

Example: */BOT_SKIP=8.*

/PAGE_LEN
Sets the length of a page in lines. The default value is 66 lines. A form-feed character will override the */PAGE_LEN* value and cause a new page to be started. DIFF needs to know the page length in order to skip headers and footers if */TOP_SKIP* and */BOT_SKIP* are specified. Also, the change summary lists changes by page and line number. For nonpaginated text, such as program source code, you should specify a value of */PAGE_LEN* greater than the number of lines in *newfile* so that the change summary line numbers will correspond to file line numbers.

Example: */PAGE_LEN=2000.*

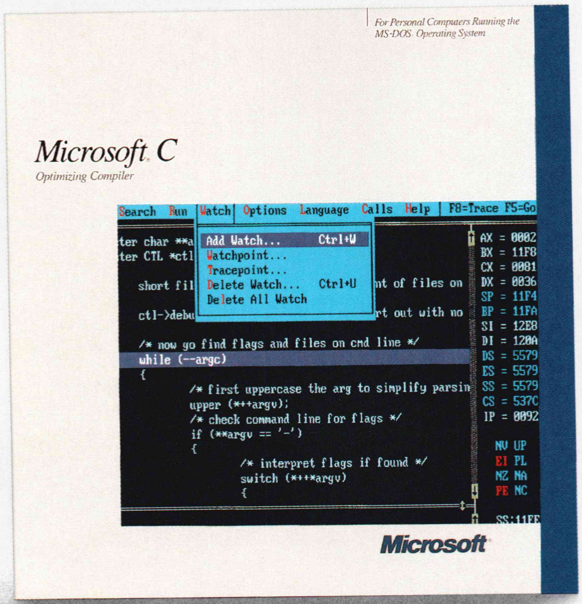
UP_CASE, /NOUP_CASE
Controls whether or not the case of alphabetic characters is significant when deciding if a line has changed. The default is */NOUP_CASE*, which means that the case of a letter is significant. This is slightly faster than */UP_CASE*.

Examples: */UP_CASE* and */NOUP_CASE*.

(continued on page 33)

Table 1: *Options*

C5.0 has three features professional programmers can't live without.



Speed.

Fast Execution Speed.

	Microsoft® C 4.0	Microsoft C 5.0
Sieve (25 iterations)	5.7	3.3
Loop	11.0	0.0*
Float	19.9	0.1
Dhrystone	22.8	19.1
Pointer	14.2	7.4

- New optimizations generate the fastest code:
  - —Inline code generation. **NEW!**
  - —Loop optimizations: **NEW!**
    - —Loop invariant expression removal. **NEW!**
    - —Automatic register allocation of variables. **NEW!**
  - —Elimination of common sub expressions.
  - —Improved constant folding and value propagation.
- Fine tune your programs for even greater speed:
  - —Coding techniques for writing the fastest possible programs are included in the documentation. **NEW!**
  - —Segment Allocation Control:
    - —Group functions into the same segment to get faster NEAR calls. **NEW!**
    - —Specify which segments receive variables to yield faster NEAR references. **NEW!**
  - —Uses register variable declarations.
  - —Mix memory models using NEAR, FAR & HUGE pointers.

Benchmarks run on an IBM® Personal System/2.™ *Time is negligible.

Speed.

Fast Compilation. Fast Prototyping.

Microsoft C Version 5.0 includes QuickC,™ which lets you edit, compile, debug, and execute in an integrated environment. It's ideal for prototyping.

- In-memory compilation at over 10,000 lines/minute. **NEW!**
- Built-in editor with parentheses, bracket and brace matching.
- Use the integrated debugger to animate through your program, add watch variables and set dynamic breakpoints. **NEW!**
- MAKE file is automatically generated for you. Simply indicate the modules you want to use, then MAKE recompiles and links only those modules that have changed. **NEW!**
- Full C 5.0 compatibility:
  - —Completely source and object code compatible.
  - —Emits CodeView®-supported executables.
  - —Identical compile/link command line switches.

standard output is a summary list showing deletions from the baseline and additions to the revision. Optionally, it can create an additional third file that is a copy of the revision file with change bars on lines that have changed or been added. No notation is added to the created file to indicate the lines in the baseline file that don't appear in the revision file.

The summary output has this form:

+pp:ll --> text

or:

-pp:ll --> text

Lines that are in *newfile* but not *oldfile* are preceded by a plus sign (+) (for "added"—get it?). Lines that are in *oldfile* but not *newfile* are preceded by a minus sign (–) (for "taken away").

The rest of the summary output is the same for added or deleted lines. The *pp:ll* portion of the output is the page number followed by the line number where the difference occurs. Added lines show page/line in *newfile*. Deleted lines show page/line in *oldfile*. *Text* is the text of the changed line. For example, if one word in one line is changed between the baseline and the new release, two summary outputs will be shown—the new line from *newfile* and the old line from *oldfile*. This feature makes it convenient to compare the differences without having to hunt for change bars. Each group of differences is separated by a blank line in the change summary output.

/RE_SYNC
Controls how many lines must match between the two files after a difference has been found before the two files are considered to be back in sync. The default is five lines. Using a larger number will make DIFF smarter when considering files that have a lot of identical lines (such as *BEGIN* or *END* statements in Pascal). Using a smaller number will make DIFF smarter when considering a file that has a lot of small changes spaced closely together. For text, a value of 2 or 3 is good. For source code, a value of 5 is pretty good.
Example: */RE_SYNC=2.*

/OUTPUT
Allows you to specify an output file for the change summary listing. In MS-DOS, this is exactly equivalent to redirecting standard output with the greater-than command-line option, and you can use either way in MS-DOS. In VMS, this matches the VMS standard redirection syntax. The default for */OUTPUT* is SYS$OUTPUT on VMS and the console on MS-DOS, but this can be redirected as a command option.
Example: */OUTPUT=FILE1.SUM.*

/BLANKS, /NOBLANKS
Lets you make blank lines (for my purposes, a blank line contains only spaces or tabs) either significant or insignificant. */NOBLANKS* is the default and means that blank lines are not considered to be significant. This is the most useful as it accounts for conditional paging and trivial source code prettying.
Examples: */BLANKS* and */NOBLANKS.*

/LOOKAHEAD
Controls how far DIFF will look forward in both files to find a rematch after it finds a difference. The default is 200 lines. A larger value lets you process files in which several pages are added or deleted between revisions. A smaller value runs much faster and uses less memory. Resynchronization time (in the general case) is proportional to the square of */LOOKAHEAD*. This value also affects the amount of memory the program uses.
Example: */LOOKAHEAD=50.*

/SKIP1
Allows you to specify a number of pages to skip in the two files before starting the compare. This is most useful when skipping tables of contents, in which the page numbers may change but nobody cares. The default for */SKIP1* is 0 pages. The */SKIP1* option sets the page-skip values for both *newfile* and *oldfile.*
Example: */SKIP1=3.*

/SKIP2
Same as */SKIP1*, except that it only affects the page-skip value for *oldfile.* Because */SKIP1* affects both files, the */SKIP2* option must appear to the right of */SKIP1* on the command line to have any effect. The two options are provided because the tables of contents may be of different lengths. There are probably other reasons why */SKIP2* needs to be here, but I can't think of any right now.
Example: */SKIP2=4.*

/TRACE
Conditionally compiled and turns on function tracing (see the main text for more on debugging options).

Table 1: *Continued*

Operation

The program's basic operation is pretty simple. It compares the two files line by line until it finds a difference. When this happens, it drops a marker in both files at the point of difference and scans ahead through both files to find where the text matches again.

When the files are resynced, the text between the point of difference and the resync point in both files is change-barred and output to the difference summary. The basic resynchronization algorithm takes only about 80 lines of code. The rest of the program is accounted for by a command-line parser and option handlers.

The major data structure used is *struct LINE*. This structure is allocated dynamically and contains a line from one of the input files in both original and uppercase form, the line and page from which the line was taken, and a link pointer used to string lines together. As the files are searched for resync, the text from the two files is chained into a linked list for each file. If blanks and headers/footers are being excluded, they are also held in the linked list while the program compares the next significant lines.
reason I never get them right the first time. There's just something about me and linked lists that causes me to leave subtle bugs in the code I write. By now, all the bugs are (hopefully) out of this code, and the code for han-

(continued from page 33)

dling the linked lists is about half its initial size. Exorcising the problems with linked lists is the reason for the *trace*, *ret*, and *ret_val* macros. If the debugging tools are compiled in, the program can display function entries and exits and the call stack on demand. (The demand call stack display only works under MS-DOS.)

A Functional Description

The function *main( )* processes the command line, opens the files, and checks for command-line errors. If no errors are found, it runs the difference check.

Function *dont_look( )* decides if a line is significant or not. If a null pointer is given to it, it returns *FALSE*, indicating that the line is significant. (This lets you factor some end-of-file logic out of later loops.) If the line comes from the header or footer area, or if the line doesn't contain printing characters and the */BLANKS* option was used, it returns *TRUE*; otherwise, it returns *FALSE*.

Function *equal( )* decides if two lines are identical. If the */UP_CASE* option was used, the uppercased lines are compared; otherwise, the original lines are compared. *Equal( )* returns *TRUE* if the lines are identical. On a small-memory system, this function could be modified to perform the uppercase conversion each time instead of carrying an uppercased copy in *struct LINE*, which would enable you to specify */LOOKAHEAD* values that were about 80 percent larger than if you left the program unchanged.

Function *position( )* is used to position the linked-list pointer to a given line in the linked list. This is used when resyncing files after a difference is found.

Function *fix( )* is included for the VAX/VMS version. The VAX C version of *fgets( )* returns a carriage return/line feed pair at the end of a line, which caused me problems detecting an end of line when inserting the change bar. The carriage return alone can't be used because of embedded carriage returns in lines that have underlined portions. *Fix( )* converts the end of line to a single line feed (new line)—but only on the VAX.

Function *index( )* searches a string for a specified character. I wrote this because of nonstandard standard library names for this function in the different compilers I use.

Function *next_line( )* reads a line from one of the input files and links it into the linked list of lines in that file. Space for *struct LINE* is allocated dynamically. It slows operation, but a conditionally compiled switch lets you look for form feeds within a line if your text might contain a form feed in any character position other than the first in a line. *Next_line( )* also keeps track of page and line numbers for the file. If the program has "looked ahead" at the file from which the next line is requested, *next_line( )* will return a pointer to the memory copy rather than reading a line from disk.

Function *discard( )* deallocates lines (allocated by *next_line( )*) that are no longer needed.

Function *vfputs( )* outputs lines to a data file. In MS-DOS, it is simply a call to *fputs( )*. In VAX/VMS, it replaces the line terminator with a carriage return/line feed pair.

Function *put( )* writes matching lines from the input file to the change-bar output file.

Function *change_bar( )* inserts a change bar into its input string. Two different algorithms are used, depending on whether the change bar is to appear to the left or the right of the text.

Function *added( )* handles lines that appear in the revision file but not in the baseline file. They are output to the change summary and, if enabled, to the change-bar file.

Error: Must specify two files
This occurs if the command line does not contain at least two file names.

Out of Memory
This error occurs if a large look-ahead is specified and/or if huge sections of text differ between the two files.

ERROR - lost sync in file <name> at page <n> line <n>
After a difference was located, DIFF could not find where the files became synchronized. To correct, increase the value for */LOOKAHEAD*. The page/line reported is the start of the point in *newfile* at which the difference was first detected.
Note: When this error occurs, the program closes any output files and exits.

Help
Usage information is printed when most command-line errors are detected.

Error: Can't open <filename>
DIFF was unable to open (for reading) one of the input files. Be sure that the file and path name are correct and that you have read privilege for that file. It may be caused by a forgotten slash (/) on an option that made DIFF interpret it as a file name.

Error: Can't create <filename>
DIFF was unable to create the optional output change-bar file. Be sure that the name is a legal one and that you have write privilege for the directory in which the name is to be placed. Could be a forgotten slash, too.

ERROR in option <option>
This error occurs when an */OUTPUT* option has a malformed or missing file name.

ERROR creating <name>
This error occurs when DIFF is unable to create an output file for the change summary. Be sure that the name is a legal one and that you have write privilege for the directory in which the name is to be placed.

Unrecognized Option: <option>
An option name is misspelled or illegal.

Error: Too many files at <name>
More than three file names appear on the command line.

Table 2: *Error messages*

The most up-to-date training in the UNIX® System, from the people who keep the UNIX System up-to-date.

What could make more sense than UNIX System training from the people who invented the UNIX System—the people responsible for all its updates and revisions. AT&T.

Created to train AT&T's own professionals, our curriculum is the most comprehensive available—including C language as well as the UNIX System. And every course is practical and job-related.

Training for everyone

- ☐ Systems developers
- ☐ Applications programmers
- ☐ Technical specialists
- ☐ System managers and users

Whatever your specialty, AT&T has the right curriculum, from basic overviews to programming to business applications and data communications. And every course is kept up-to-the-minute with such recent advances as System V Release 3.0.

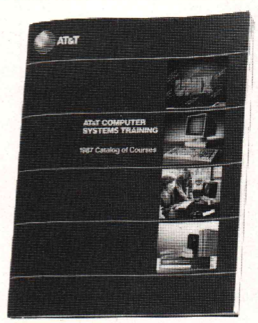
Individual attention

Classes are limited in size, so that each student can be given individual instruction and supervision. In laboratory classes, each student is assigned his own terminal. Instruction is by AT&T UNIX System veterans and is personal, thorough, productive.

Classes forming now

Reserve as quickly as possible for preferred dates at our completely equipped training centers in Atlanta, Chicago, Dublin, OH, Los Angeles, Princeton, NJ, and Sunnyvale, CA. Or we'll arrange instruction on your site at your convenience. But don't wait—call or write now for information and seat reservations.

Free fact-packed training catalog:
Call 1 800 247-1212, ext. 1002, or mail this coupon.



Registrar, AT&T Training, **DDJ 8/87**
P.O. Box 45038, Jacksonville, FL 32232-997
Please rush me your course catalog with information on:
☐ UNIX System training ☐ UNIX System video training
☐ Data communications and networking training

Name ____________________
Title ____________________
Company ____________________
Address ____________________
City ____________ State ______ Zip ________
Phone (   ) ____________________

AT&T
The right choice.

CIRCLE 396 ON READER SERVICE CARD

WHAT'S THE DIFF?
(continued from page 34)

Function *deleted( )* handles lines that appear in the baseline file but not in the revision file. They are output to the change summary.

Function *resync( )* is the guts of the program—it resynchronizes the two files after they get out of sync. The input arguments are pointers to the lines that are found to differ in the two files. It works in the following manner.

The first line in the revision file that doesn't match the baseline file is compared to lines in the baseline file until a matching line is found or you have looked at */LOOKAHEAD* significant lines, whichever comes first. If you find a matching line in the baseline file, you compare the next */RE_SYNC* lines to ensure that they too match. If so, you consider yourself synced, print the change information, and exit with the file input pointers at the first lines that match again. If */RE_SYNC* lines don't match, you continue the search.

If you look at */LOOKAHEAD* lines in the baseline file without matching the revision file from the point of difference, you move ahead one line in the revision file and then repeat. Eventually, you'll either find a match or you'll have moved ahead */LOOKAHEAD* lines in the revision file. At this point, you give up and exit from the program.

This is a fairly brute-force method as a file that contains one or two large difference sections and a large number of small differences will perform poorly because */LOOKAHEAD* needs to be large enough to accommodate the big differences but will be inefficient on the small differences. It should be relatively easy to make this adaptive by starting with a small value for */LOOKAHEAD*, and when a large difference is encountered, at label *no_sy* pushing the old value of */LOOKAHEAD*, setting a larger value (say, double the old value), and calling *resync( )* recursively. This way, given enough memory, the program will always resynchronize.

Function *diff( )* handles the printing of lines that match and calls *resync( )* when differences are found.

Function *page_skip( )* skips the front ends of files when the */SKIP* options are used.

Function *help( )* prints the usage summary when command-line errors are detected.

Function *open_files( )* opens the two input files and, if specified, the change-bar output file.

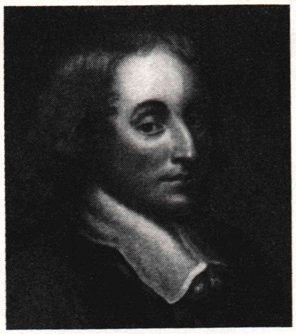
Function *redirect( )* redirects the VAX standard output. Because this program has a variable number of arguments, it's easiest to use if installed as a foreign command, and the standard redirection doesn't work then. Incidentally, to install this, use the following command:

```
DIFF :== $ diskname:[pathname]
                          DIFF.EXE
```

Redirect( ) works under MS-DOS as well, but it's not required.

Function *strip_opt( )* parses the command line. It is designed around the VMS command syntax, which I like better than MS-DOS or Unix (at least for options). I get annoyed when I can't abbreviate options I use interactively or leave them spelled-out in command scripts. I also get annoyed when options are case sensitive, especially if some of the options are lowercase, some are uppercase, and some are mixed (as in a certain C

What Pascal did for his dad, we can do for you.



In 1644, 19-year-old Frenchman Blaise Pascal devised one of the first mechanical calculators to help his father with time-consuming tax computations. Called the Pascaline, it was the size of a shoe box and used numbered dials to process numbers up to 999,999,999. Pascal's idea represents what Lifeboat is all about: Developing tools that enable you to do your job faster and better than ever—and help you make your own mark on history. LIFEBOAT

Get more pluses with new ADVANTAGE C++.

ADVANTAGE C++™

Expand your programming capabilities with C++, the object-oriented language from AT&T that gives you all the benefits of C without its limitations. Our enhanced ADVANTAGE C++ is the only full C++ implementation available.

- Develop large and complex programs with greater resilience, fewer bugs.
- Fully compatible with your existing C programs and libraries.
- Code is more reliable and maintainable.
- Fully tested and documented.
- Small to large memory models.
- Microsoft Windows compatible.
- Available for Microsoft and Lattice C compilers: MS-DOS and XENIX operating systems.

Now your programs can do more than one task at once.

TimeSlicer

TimeSlicer is a linkable library of C functions to create multitasking and real-time programs at the application level rather than interfacing with the operating system.

- Highly efficient — 10,000 context switches/second; 80 micro seconds interrupt latency.
- Unlimited number of tasks can be run concurrently.
- Create, suspend or terminate tasks at run-time.
- Supports large and small memory models; preemptive and non-preemptive modes; waking-up of tasks to optimize special event processing; and interrupt service routines written in C.
- Extensive intertask communication capability.
- Compatible with Lattice C, Microsoft C, ADVANTAGE C++ and assembly language.

Design advanced graphics for today's applications.

ADVANTAGE Graphics™

ADVANTAGE Graphics gives you the expanded graphics capabilities and functionality required for today's sophisticated programming needs.

- Easily create multi-window and other object-oriented applications.
- Realistic, typeset-style graphics.
- Access all levels of the graphics system.
- Full and complete clipping of all graphic elements and text.
- Predefined and user-defined line styles, fill patterns and cursors.
- Create and instantly switch between multiple on- or off-screen graphic windows.
- Supports global, local and virtual coordinate reference frames.

ADVANTAGE 386 C™ and ADVANTAGE 386 Pascal™—Two compilers generating 32-bit protected-mode code for the 80386 under MS-DOS.

ADVANTAGE Disassembler™ — Lets you disassemble object code into source code and make changes to fit your needs.

ADVANTAGE Make™—The best-documented MAKE utility available; feature-packed MS-PC/DOS version of UNIX MAKE facility.

PANEL Plus — A powerful, interactive tool that makes screen design as easy as writing with a word processor.

QuickScreen™ — A full-featured, time-saving screen design program that requires no written code.

RUN/C Professional—Powerful C interpreter featuring loadable libraries and advanced debugging. Now with Microsoft 4.0 compatibility.

Call 1-800-847-7078
In NY: 914-332-1875
or see your local
Lifeboat Affiliated Dealer

LIFEBOAT

55 South Broadway, Tarrytown, NY 10591, Telex #510-601-7602

INTERNATIONAL SALES OFFICES

Australia/New Zealand:
MOS Computer Software/
Charlton Distributors
Auckland (09) 766-361
Canada: Scantel Systems
Toronto (416) 449-9252
Denmark: Ravenholm
Copenhagen 288-7249

England: Grey Matter, Ltd.
Devon 364-53499
System Science, Ltd.
London (01) 248-0962
France: Compusol
Paris 14 530 0737
Italy: Lifeboat Associates Italia
Milan 02-464601

Japan: Lifeboat, Inc.
Tokyo 03-293-4711
SATT Software
Tokyo 03-295-3390
Netherlands: SCOS Automation BV
Amsterdam 020-10 69 22
Spain: Micronet, S.A.
Madrid 1-262-3304

Switzerland: Euro-Link
Willisau 4145 813 514
West Germany:
MEMA Computer GmbH
Frankfurt 069-347226
Omnitex
Rheinfelden 07623/61820

CIRCLE 359 ON READER SERVICE CARD

On April 2, 1987 IBM and Quarterdeck announced the next generation in personal computing.

Introducing DESQview 2.0. Improving the past and ready for the future right now.

In one sweeping announcement from Miami Beach and New York City, IBM established new standards of performance for personal computers, with its new Personal System/2.™ Quarterdeck was there with IBM and simultaneously helped establish new standards for multi-tasking and multi-windowing.

We were there for them then. We're here for you now.

If you use two or more software programs, if you use a PC-compatible machine, if you own a new 80386 computer, if you've just bought one of the new Personal System/2 computers, or if you've tried Microsoft Windows and were disappointed but still need the power of graphics programs, DESQview 2.0 is the answer.

Consider this. InfoWorld voted DESQview's earlier version 1986 Product of the Year. SoftSector gave it the Editor's Choice Award. In PC Tech Journal's "System Builder Contest" at Comdex Fall it was voted best operating environment. And 450,000 dedicated users on four continents have voted yes with their dollars.

The new DESQview 2.0 is an order of magnitude better.

This unique software program enhances the power of your personal computer and makes it more convenient to use. It still gives your PC the power of many PCs. It still does windows. It still multi-tasks. It still breaks the DOS 640K barrier. It still transfers data. It still dials your phone. It still gives you menus for DOS. It still remembers your keystrokes (macros). It still runs your existing programs and your new programs soon to come. In fact now you can even run Windows-, GEM-, and Topview-specific programs too. And with 386 machines and our Expanded Memory Manager it still becomes a 386 control program, but now you can run text *and* CGA graphics programs in background.

The new DESQview 2.0.
For us it's the next logical step.
For you it's windows of opportunity.

SYSTEM REQUIREMENTS
• IBM Personal Computer and 100% compatibles (with 8086, 8088, 80286 or 80386 processors) with monochrome or color display; IBM Personal System/2 • Memory: 640K recommended; for DESQview itself 0-145K • Expanded Memory (Optional): expanded memory boards compatible with the Intel AboveBoard; enhanced expanded memory boards compatible with the AST RAMpage • Disk: Two diskette drives or one diskette drive and a hard disk • Graphics Card (Optional): Hercules, IBM Color/Graphics (CGA), IBM Enhanced Graphics (EGA), IBM Personal System/2 Advanced Graphics (VGA) • Mouse (Optional): Mouse Systems, Microsoft and compatibles • Modem for Auto-Dialer (Optional): Hayes or Compatible • Operating System: PC-DOS 2.0–3.3; MS-DOS 2.0–3.2 • Software: Most PC-DOS and MS-DOS application programs; programs specific to TopView 1.1, GEM 1.1 and Microsoft Windows 1.03 • Media: DESQview 2.0 is available on either 5¼″ or 3½″ floppy diskettes

Rush me DESQview 2.0! Today!

No. of Copies	Media 3½″/5¼″	Product	Retail Price ea.	Total
		DESQview 2.0	$129.95	$
		Shipping & Handling USA / Outside USA	$ 5.00 / $ 10.00	$
		Sales Tax (CA residents)	6.5%	$
			Amount Enclosed	$

Payment: ☐ Visa ☐ MC ☐ AMEX ☐ Check

Credit Card: Valid Since ____/____ Expiration ____/____

Card Number: ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Credit Card Name ____________

Shipping Address ____________

City ______ State ______ Zip ______ Telephone ______

Mail to: Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405
NOTE: If you own DESQview call us for a special upgrade offer, or send in your DESQview registration card and we'll send you upgrade information.

DDJ 8

DESQ view™
Quarterdeck

Quarterdeck Office Systems • 150 Pico Boulevard, Santa Monica, CA 90405 • (213) 392-9851

CIRCLE 284 ON READER SERVICE CARD

WHAT'S THE DIFF?
(continued from page 36)

compiler made by a company known for PC operating systems and mice). But then, we software types are known for our unreasonable likes and dislikes—I still use WordStar—and this parser can easily be made more Unix-like by changing the *OPT_FLAG* define and the literal arguments to *match( )*.

Function *upper( )* converts its string argument to all uppercase letters.

Function *match( )* checks for (possible) partial matches of command-line option strings with a pattern string. To make this really bulletproof, it ought to check for a minimum number of matching characters. Right now it doesn't, but this hasn't caused any trouble so far.

Function *num( )* retrieves the value parameter from command options.

The rest of the program is conditionally compiled and is strictly debugging support for making modifications. Whenever I find occasion to tweak the program, it seems to die silently the first couple of times I run it. With the debugging support in, if you run the program with the */TRACE* option, it will print a message each time it enters or exits a function. Pressing T toggles tracing on and off. Pressing S displays the current call stack. This is a great help in finding where the program is hung in a loop. Be warned: it's also hours of fun to watch DIFF crunch a 200-page document with */TRACE* on.

Areas for Enhancement

Because DIFF has solved my immediate change-bar concerns, it's unlikely I'll be making any major enhancements in the future. I am releasing DIFF into the public domain, however, and would appreciate hearing from those of you who make modifications and improvements to the program. And I have two suggestions to start you off.

The first major enhancement I can see would be useful in archiving versions of source code. DIFF could be modified to emit line editor script files that contain the commands to transform one version of a file into another. This would let you keep only the first version of a file in its complete form. Subsequent versions would be kept as differences (the editor script file), so any version could be recreated by transforming the original into the desired version with a series of editing operations. Many configuration management/source code control tools use this type of system to save disk space.

The second major enhancement is more difficult and possibly is useful only to a small number of people. The current version of DIFF is line-oriented, and thus it can be fooled by any changes in format—for example, reparagraphing a document with different margin settings, adding several words for a paragraph (and thus the paragraph to be reformatted), and common alterations to source files such as tabbing/detabbing or pretty-printing.

It's rare that reformatting effects are major problems, but if they become a concern, DIFF could be modified to act on tokens rather than on lines. The major headache in tokenizing is that the lexical rules are different for program source and documents. A less serious problem is relating the change information from the token stream back to the line-oriented source text. Compilers seem to be able to do this, so several elegant solutions probably exist.

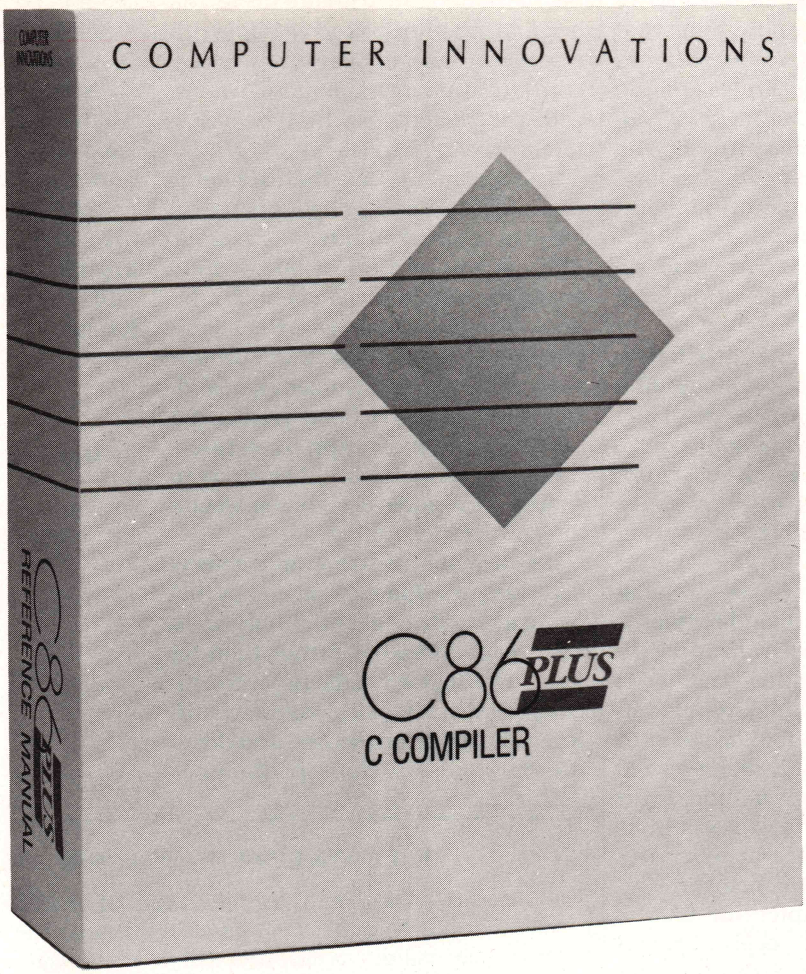
Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listing begins on page 66.)

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**



100% ROMABLE

C86PLUS™ C compiler and ROMpac™ Development Tool

(see other side for details)

CIRCLE 96 ON READER SERVICE CARD

MANY CLAIM IT. FEW PRODUCE IT.

That's why many professional C programmers choose Computer Innovations' C86PLUS C compiler and the ROMpac Development Tool to create stand alone ROM applications.

FAST EXECUTING PROGRAMS

C86PLUS supplies the power to make your programs run faster because it is based on a proprietary compiler design technology that applies artificial intelligence to produce highly optimized code.

BENCHMARKS (seconds)	C86PLUS™ v.1.07	Microsoft C v.4.0	Lattice C v.3.1
sieve.c (compact mod.) Eratosthenes	21	29	30
rsieve.c (large mod.) (registers)	14	17	25
loop.c (medium mod.) (loop optimization)	1	28	45
randio.c (small mod.) (random access I/0)	10	21	24
tlong.c (small mod.) (long integer math)	128	155	198
tdbl.c (small mod.) (floating point math)	38	37	47

Run on an IBM PC-AT; 8MHZ; 512 K RAM; 20Mb hard disk; PC-DOS 3.1.

C LIBRARY SOURCE CODE

Computer Innovations ships C library source code and run-time start-up source code free of charge, with no royalties attached. The **C86PLUS** library source code can be used for customizing and fine tuning library routines, and is often critical in developing embedded systems applications.

INTERRUPT ROUTINES IN C

C86PLUS allows you to write interrupt service routines in C rather than assembler. This assures portability should your compiler version change and makes it easier to maintain and modify existing C code. It also minimizes your initial development time because a full working knowledge of the 8086 assembler is not required.

TELEPHONE TECHNICAL SUPPORT

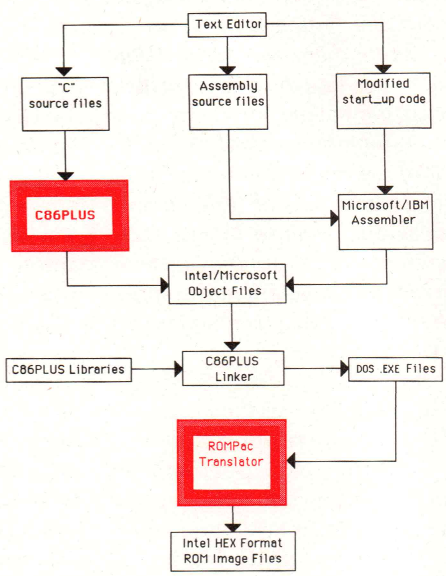
Computer Innovations stands behind its products by offering timely and intelligent technical support. Our technical support staff is available by phone (201 542-5920 Monday through Friday from 10 a.m. to 5 p.m. EST.

ROMpac™

ROMpac is a high-level software development package designed to assist C programmers who need to compile C programs and burn the executable program into ROM.

ROMpac is intended for use in an MS-DOS environment for the creation of stand alone ROM programs that execute in dedicated machines. Typical uses include: **data acquisition systems, communication controllers, and process control systems.** The common factor is that one program is in complete control of the CPU. There is no limit on the size of the program, other than physical constraints imposed by the 8086 architecture.

HOW "ROMpac" WORKS



SYSTEM REQUIREMENTS: CPU: Intel 8086/186/286/386; Operating System: MS/PC-DOS 2.1 +; RAM: 512K; Hard Disk required.

ORDER NOW:

C86PLUS w/ROMpac Development Tool $699
C86PLUS C Compiler $497
ROMpac Development Tool $250

800-922-0169

Purchase C86PLUS w/ROMpac and receive one-year of compiler updates. Call for details.

COMPUTER INNOVATIONS, INC.
980 SHREWSBURY AVENUE, TINTON FALLS, N.J. 07724 U.S.A.
(201) 542-5920 (800) 922-0169 TELEX 705127 COMP INNOV UD

Optimizing Compilers for C

by Richard Relph

You've seen the ads: Datalight challenges Microsoft. Our C compiler expert Richard Relph saw the ads and sent for Datalight's compiler. What he found when he began to test it must have given him mixed feelings. For the past two years Richard has been involved in developing the DDJ suite of benchmarks for C compilers. The Datalight compiler flattened those benchmarks, making them worthless. What has made our benchmarks obsolete and raised the stakes for C compiler vendors is something called global optimization, common on mainframe and minicomputers and now coming to the desktop. Because it was Richard's work that was made obsolete, we gave him the assignment of reporting on the optimization techniques that did the job.—eds.

This article describes some of the optimizations that a C compiler can perform to make the resultant code either smaller or faster. For many languages, such as FORTRAN, optimization is necessary because the language is poorly matched to the target processor. But C's close match to underlying target instructions and its rich set of operators have made optimization largely optional for C code. A well-written C program, compiled with a nonoptimizing compiler, can perform favorably compared to the same program written in FORTRAN or Pascal.

Richard Relph, 846 Salt Lake Dr., San Jose, CA 95133. Richard is a software and hardware consultant. He has written compilers and embedded systems.

Function range optimization is the biggest step forward.

Nevertheless, C code can be optimized, for some applications it may be highly desirable for the compiler to do the optimization, and optimizing C compilers have existed for some time for non-Intel processors. Tartan Labs and Green Hills come to mind when thinking about optimizing C compilers for VAXs or 680x0s.

With the apparent glut of C compiler suppliers vying for the MS-DOS market, it was only a matter of time before one of them decided to step above the crowd and provide a reliable optimizing C compiler. Datalight beat all others to the punch by delivering such a compiler in February of this year. Computer Innovations has also shown a product that it claims is optimizing, and others will announce and deliver products before the year's end.

Datalight provided me with its Optimum-C package in February, and it is this compiler upon which this article is largely based.

Varieties of Optimizations

I have used the term *optimization* rather freely, but so do compiler vendors. There are, in fact, several kinds of optimizations, and I would like to distinguish what I mean by optimization from what other people (particularly compiler vendors) might mean.

Basically, optimization can occur at two places in the compilation process. The simplest, so-called peephole, optimizations occur in the final stages. A peephole optimizer can eliminate various dumb-looking instruction sequences. It can get rid of redundant loads (loading a register with a value that it already contains) and some jumps around jumps (depending on the target instruction set and the knowledge the compiler maintains about function size). Some peephole optimizers do more advanced things.

The other place in the compilation process in which optimizations can occur is in the middle stages. This is after the program has been "read" by the compiler and converted to some intermediate form. In many compilers this intermediate form is both host and target independent, thereby making optimizations at this stage very useful if the compiler must support multiple targets.

These intermediate optimizations, which are more interesting for my present purposes, can occur over five ranges, which I refer to here as statement, block, function, module, and program. These range designations are pretty self-explanatory. A statement is a statement. A block is a sequence of instructions in which there are no jumps or labels used as entry points. A function is a function. A module is a module or source file. Optimizations over all these ranges can be performed using current compilation models—edit, compile, link, and debug. Optimizations over the last range—program—cannot. In this case the compile and link phases

must be combined.

Optimization from peephole through statement range is nothing new. These optimizations have been done for a long time by almost all the C compilers I am familiar with (that's a lot). Some examples of statement range optimization are constant folding, strength reduction, and dead statement elimination. These are basic, and I will not discuss them here.

Block range optimization is where the subject begins to get interesting. Block range optimizations are done by some of today's better PC-based compilers. The most common and important block range optimization is common subexpression elimination, and you can see it in practice later in this article.

But it is function range optimization (sometimes called "global optimization") that represents the biggest step forward in C code optimization today. Function range optimization is really what this article is about, so I'll defer examples and descriptions for a few lines while I dismiss module and program range optimization.

Module range optimization is rare. The only commercial compilers I know of that do this are those from Tartan Labs. An optimizer dealing with the module range turns some small functions into in-line code or automatically passes arguments to static functions in registers. This is called "interprocedure analysis." In fact, all the simpler kinds of optimizations can be performed across several functions, and when this is done, it is module range optimization. As I say, though, this is rare.

The last range for optimization is program range optimization. This takes you beyond rare and to nonexistent. When some compiler (and built-in linker) implements program range optimization, you can begin to say things such as "this is optimal" or "as good as the best assembly language" because all the information about the program is available to the compiler at one time. This is still a dream, but some companies are discussing this kind of compiler. If you want to be prepared, get a machine with lots of memory, a superfast processor, and, perhaps most important, an uninterruptible power supply, because this compiler's compilation time will be measured in hours to weeks instead of seconds to hours.

Table 1, below, summarizes the availability of these different optimizations in present-day compilers. The optimization ranges are not as firm as they appear to be from the preceding outline. Compiler vendors may choose to implement only one or two of the simpler function range optimizations—not the full deck. Companies currently doing this are Wizard/Borland and MetaWare, which implement the simplest automatic register allocation scheme, and MetaWare and Microsoft, which implement cross-jump optimizations. Many compiler vendors implement some sort of switch statement optimizer, which is hard to categorize (I believe it is a block range optimization). But none of the vendors, until Datalight, has attempted a reasonably complete function range optimizer.

Modern Optimization Techniques

The rest of this article provides a tour of the major function range optimizations. I give each a name (note that I did not say "the" name), a brief description of the optimization, and a sample code fragment to which the optimization applies. I'll start with some of the more basic and simpler optimizations and work toward the more advanced.

Constant Propagation

Constant propagation is used when a variable has a constant value over a portion of the function. Although constant propagation is not very useful by itself, in conjunction with common subexpression elimination and invariant code motion (discussed later), it becomes important.

For the next several optimizations, I will refer to the following simple code segment:

```
func( p )
        int p;
{
        int i;
        int j;

        i = 5;
        for (j = 0; j < i; j++)
                ;
}
```

which, after constant propagation, becomes:

```
func( p )
        int p;
{
        int i;
        int j;

        i = 5;
        for (j = 0; j < 5; j++)
                ;
}
```

As this example shows, constant propagation may create dead assignments. The assignment $i = 5$ becomes pointless unless i is used offstage somewhere.

Copy Propagation

Copy propagation is like constant propagation, except that the compiler keeps track of which variables hold the same values rather than noting that a certain variable holds a constant value. This results in substitution of one variable for another when they have the same value. The possible advantage this gives you is that one of the variables may be faster to get to (because it is in a register) than the other.

Stage	Range	Rarity	Example
final		common	"peephole" optimization
intermediate	statement	common	dead statement elimination
intermediate	block	some PC compilers	common subexpression elimination
intermediate	function	new to PC compilers	code hoisting
intermediate	module	rare	interprocedure analysis
intermediate	program	nonexistent	

Table 1: *Use of various optimizations in present-day compilers*

(continued from page 43)

In the preceding example, if *i* were assigned the value *x* instead of the constant *5*, then copy propagation would apply and you would see the following:

```
i = x;
for (j = 0; j < x; j++)
        ;
```

As is the case with constant propagation, copy propagation may create some dead assignments; here, if *i* is not used subsequently, the assignment *i = x* is dead.

Dead Assignment Elimination

Dead assignments are assignments to variables that are not used before the variables are assigned again. In the constant propagation example, after the constant has been propagated, the assignment to *it* is useless, or dead. Such an assignment can be eliminated. After dead assignment elimination in the example you have:

```
func( p )
        int p;
{
        int i;
        int j;

        /* i = 5 */
        for (j = 0; j < 5; j++)
                ;
}
```

Dead assignment elimination may in turn result in a dead variable, as has happened in this example. Variable *i* is now dead and can be eliminated, which leads me to the next subject.

Dead Variable Elimination

A dead variable is a variable that is never referenced. Looking again at the constant propagation example, after constant propagation and dead assignment elimination, the variable *i* may no longer be needed, so its space on the stack or in a register can be freed for other use.

Eliminating it, you get:

```
...
{
        /* int i; */
        int j;

        /* i = 5 */
        for (j = 0; j < 5; j++)
                ;
}
```

The variable *p* is also dead, but being an argument to the function, it is not removable.

Dead Code Elimination

As you can eliminate dead assignments and dead variables, so too can you eliminate dead code. Dead code is any code that can never be reached. Although dead code is rare in practice, this optimization is fairly easy to do, so why not? Many compilers implement simple forms of this optimization even though they do

```
struct x {
        int i;
        char c;
} d[ 10 ][ 10 ], s[ 10 ][ 10 ];
copy()
{
        int i, j;
        for (i = 0; i < 10; i++)
                for (j = 0; j < 10;
j++)
                        d[ i ][ j ] =
s[ i ][ j ];
}
```

Example 1: *C code to be optimized*

not implement other function range optimizations. For the remaining optimizations, we will use the code segment in Example 1, page 44.

Global Common Subexpression Elimination

Many functions use and reuse subexpressions in the computation of complete expressions. Such subexpressions are said to be common. If a compiler can detect such subexpressions, compute them once, save the result, and simply refer to the saved result, recomputation can be avoided. This is particularly important with floating-point and other compute-intensive data types. Note that the compiler may create a variable in the process.

The following shows the code in Example 1 after common subexpression elimination.

```
. . .
{
        t0 = i * 10 + j;
        d [0] [ t0 ] = s [0] [ t0 ];
}
. . .
```

Lifetime Analysis

Lifetime analysis is the first of the hard optimizations. What lifetime analysis attempts to do is determine which variables have meaningful values over what range of the function.

Variables that have nonoverlapping ranges may share processor resources, especially registers. For straight code it is easy to see how to do this analysis, but loops and *gotos* make it much harder.

Register Allocation

After the lifetime of each variable is determined, important variables can be identified. Important variables are those that are referred to often, either because they are named frequently or because they occur inside loops. There is usually a multiplier applied to the "reference count" obtained for variables in loops. So, once the compiler has ranked the variables in importance, its goal is to use the processor's resources well. A well-known technique for doing this (used by Datalight) is called "coloring" because of its similarity to the map-coloring problem—except your "map" is a variable usage graph.

The map-coloring problem is this: Given a fixed number of colors (CPU registers), color the map (the variable usage graph) so that the fewest (preferably 0) number of states (variables) are left uncolored (not in registers) assuming that no two adjoining states (variables with overlapping lifetimes) have the same color (register).

A much simpler allocation strategy is used in some existing compilers. These compilers merely take the most important variables and dedicate them to registers throughout the function.

Loop Invariant Code Motion

Similar to common subexpression elimination, loop invariant code motion (sometimes called "code hoisting") notices that some subexpressions are not affected by the execution of the loop. Because most loops are executed more than once, such subexpressions are logically common (refer to the definition under common subexpression elimination). By computing them once before the loop is entered, the compiler can save a lot of run-time recomputations.

After one level of code motion, our example looks like this:

(continued from page 45)

```
. . .
{
        t0 = i * 10;
        for (j = 0; j < 10; j++)
        {
                t1 = t0 + j;
                d [0] [ t1 ] = s [0] [ t1 ];
        }
}
. . .
```

Loop Induction

What loop induction is, conceptual-

What loop induction is, conceptually, is strength reduction on loops. If a loop has a subexpression in which one part is loop index sensitive and the operator is multiply, it is possible to replace the subexpression by a variable that gets added to for each change in the loop index. The remaining examples point out the usefulness of this technique (particularly when such constructs may be present but not obvious) and give some further sense of how function range optimizations can be used.

ous dead eliminations, we have:

```
. . .
for (t0 = 0; t0 < 100; t0 += 10)
        for (j = 0; j < 10; j++)
        {
                t1 = t0 + j;
                d [0] [ t1 ] = s [0] [ t1 ];
        }
. . .
```

This code can be optimized further, though. Here it is after loop induction on *t1*:

```
. . .
for (t1 = t0; t1 < t0 + 10; t1++)
        d [0] [ t1 ] = s [0] [ t1 ];
. . .
```

Here it is after common subexpression elimination on the implied multiply in the loop:

```
. . .
for (t1 = t0; t1 < t0 + 10; t1++)
{
        t2 = t1 * sizeof( struct x );
        ((char *)&d) + t2 = ((char
                             *)&s) + t2;
}
. . .
```

The following code fragment shows what the effect of reinduction on *t1* and *t2* will be:

```
. . .
for (t1 = t0 * sizeof( struct x );
        t1 < (t0 + 10) * sizeof( struct
                                    x );
        t1 += sizeof( struct x ))
{
                ((char *)&d) + t1 =
                        ((char *)&s) + t1;
}
. . .
```

Here it is after more invariant code motion:

```
. . .
t3 = (t0 + 10) * sizeof( struct x );
for (t1 = t0 * sizeof( struct x );
        t1 < t3;
        t1 += sizeof( struct x ))
. . .
```

The machinations I have just been through can be expected to yield space and time benefits in the neighborhood of 30 percent. Datalight has improved its dhrystone performance from 1,084 to 1,284 dhrystones

per second through these kinds of optimizations.

Summary

Although the discussion in this article has been based largely on work with one optimizing compiler, more optimizing compilers will be coming out soon. One problem they will present to software developers has to do with naming. There is no agreed-upon name for many of these optimization techniques. Just because some vendor says it has "xyz" optimization doesn't mean nobody else does; it may just mean that nobody else calls it xyz. I hope this article has provided you with some means to understand vendors' claims and counterclaims and to make an informed choice.

I also hope I have given you a sense of the importance of this development in personal computer compiler technology. The optimizations I have discussed here are considered basic by minicomputer and mainframe compiler standards, and it is precisely because of the lack of such "basic tools" that many computer professionals consider personal computers to be toys. Well, these particular basic tools have arrived. I think we can now safely put the "toy" complaint to rest, and I firmly believe that, when program range optimizers arrive, personal computers will be among the first machines of any size to employ them.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 4.**

The Leaders Made PVCS
The Leading Source Code Control System.

NOW VAX/VMS & MS-DOS Versions

When it comes to maintaining their most valuable asset, the leading software publishers rely on the POLYTRON Version Control System (PVCS). From accounting firms to airlines, the leading service companies depend on PVCS to maintain the integrity of their programs. Leading manufacturing companies use PVCS to maintain their state-of-the-art software. Leading high technology companies turn to PVCS to handle configuration management for software projects that represent an investment of hundreds of thousands of dollars. The largest aerospace companies and defense contractors use PVCS to maintain integrity of projects during development and after delivery of software. Independent programmers use PVCS to improve their productivity and software quality for themselves and their clients.

Simplify Configuration Management

When large and complex software programs are being developed on personal computers or VAX minicomputers, effective management of the revisions and versions becomes critical. PVCS simplifies this process and lets you effectively control the proliferation of code changes. We used UNIX SCCS and RCS as models. However, our own experience, and the input of hundreds of programmers and managers has enabled us to significantly improve upon these models.

PVCS provides many powerful functions including:

- Storage & Retrieval of multiple revisions of text.
- Maintenance of a complete history of changes.
- Maintenance of separate lines of development using branching.
- Merging simultaneous changes.
- Resolution of Access Conflicts.
- Modules can be retrieved by their own revision number, system version name, or specified date.
- Uses "reverse deltas" to rebuild a prior version making PVCS the fastest version control system over the project life cycle.
- Projects already under development or in the maintenance stage can be easily put under the control of PVCS.

Manages Development On Local Area Networks

Programming teams using Local Area Networks depend on PVCS to help the managers and team members work together. In fact, Novell and 3Com themselves depend on PVCS to manage the versions of their own network software products.

Supports MS-DOS and VAX/VMS Development

Now, companies that develop software on VAX systems running VMS can also use PVCS. And since the VMS and MS-DOS versions of PVCS use the same "logfile" format, you can easily develop software on PCs and maintain the code on the VAX or vice versa. The menu-driven, screen-oriented interface (and optional command-driven interface) makes it easy for programmers and librarians or administrators to use PVCS on a PC or VAX or both systems.

PVCS Maintains System Integrity

PVCS prevents corruption of code that could ordinarily result from security breaks, user carelessness or malfunctions. The levels of security can be tailored to meet the needs of your project.

PVCS & PolyMake Work Together

PolyMake, the leading MS-DOS make utility, is now available for the VMS operating system. This allows you to write makefiles that will function in both PC and VAX environments. Additionally, PolyMake reads time & date stamps of PVCS archives for fast, accurate program rebuilding.

PVCS and PolyMake Maintain Source Code Written In Any Language.

The Preferred Version Control System

The customers listed below are just a few of the innovative leaders that have made PVCS the leading version control program for personal computers.

Alcoa Aluminum
Arthur Anderson
AT&T
Ashton-Tate
Bank of America
Bell Labs
Bendix
Boeing
CIGNA
Citibank
3Com
Colonial Penn
Commerce Clearing House
Control Data Corp.
Corvus
CXI
Digital Equipment Corp.
Deloitte Haskins + Sells
Diebold
Dow
Dunn & Bradstreet
EDS
Educational Testing Service
E-Systems
Equitable Life
Federal Express
First Boston
Ford
Fox Software
Fujitsu
GTE
Hardees
Hewlett-Packard
Honeywell
Hughes Aircraft
IBM
Industrial Networking
Intel
ISC Aerospace
IVAC
Javelin
Lattice
Lawrence Livermore
Lotus
McData Corp.
McDonnell Douglas
Mead Data Central
MIT Lincoln Labs
Nastec
Novell
NCR Technologies
Pitney Bowes
Plexus Computers
Price Waterhouse
ROLM
Rockwell International
Safeco
Sears
Security Pacific
Sperry
Software Publishing
Spacelabs
Standard Oil
Standard & Poors
Tandem
Tektronix
Telex
Texas Instruments
Touche Ross
Unisys
United Airlines
United Parcel Service
United Technologies
U.S. West
Westinghouse Electronics
Xerox

Only PVCS meets the needs of independent programmers and corporations. Once you standardize on PVCS, the archives used to track and monitor changes are interchangeable between any PVCS product. You will receive full credit for your initial purchase if you upgrade to a higher-priced MS-DOS version of PVCS.

Personal PVCS — Offers most of the power and flexibility of Corporate PVCS, but excludes the features necessary for multiple-programmer projects.

Corporate PVCS — Offers additional features to maintain source code of very large and complex projects that may involve multiple programmers. Includes multi-level branching to effectively maintain code when programs evolve on multiple paths (e.g. new versions for different host systems, or a new program based on an existing program).

Network PVCS — Extends Corporate PVCS for use on Networks. File locking and security levels can be tailored for each project.

PVCS for VAX systems — Requires VMS. Uses the same interface and archive format as MS-DOS version. Supports branching and offers file locking and other security features for multiple-programmer projects.

	MS-DOS*	VMS		
	PC/XT/AT	Micro VAX II	VAX 7xx	VAX 8xxx
Personal PVCS	$149			
Corporate PVCS	$395			
Network PVCS	$995**	$4,950	$9,500	$10,500+
PolyMake	$149			
Network PolyMake	$447**	$1,250	$2,375	$2,500+

*Compatible with MS-DOS 2.0 through 3.3. Compatible with the IBM PC/XT/AT & other MS-DOS PCs.

**5 Station LAN License. Call for pricing on larger Networks.

TO ORDER:

VISA/MC 1-800-547-4000.
Dept. No. 355.
Oregon & Outside USA call (503) 645-1150.
Send Checks, P.O.s to: POLYTRON Corporation, 1815 NW 169th Place, Suite 2110, Beaverton, OR 97006.

POLYTRON

High Quality Software Since 1982

SAS Institute Inc. Announces

Lattice C Compilers for Your IBM Mainframe

Two years ago...
SAS Institute launched an effort to develop a subset of the SAS® Software System for the IBM Personal Computer. After careful study, we agreed that C was the programming language of choice. And that the Lattice® C compiler offered the quality, speed, and efficiency we needed.

One year ago...
Development had progressed so well that we expanded our efforts to include the entire SAS System on a PC, written in C. And to insure that the language, syntax, and commands would be identical across all operating systems, we decided that all future versions of the SAS System—regardless of hardware—would be derived from the same source code written in C. That meant that we needed a C compiler for IBM 370 mainframes. And it had to be good, since all our software products would depend on it.

So we approached Lattice, Inc. and asked if we could implement a version of the Lattice C compiler for IBM mainframes. With Lattice, Inc.'s agreement, development began and progressed rapidly.

Today...
Our efforts are complete—we have a first-rate IBM 370 C compiler. And we are pleased to offer this development tool to you. Now you can write in a single language that is source code compatible with your IBM mainframe and your IBM PC. We have faithfully implemented not only the language, but also the supporting library and environment.

Features of the Lattice C compiler for the 370 include:

- **Generation of reentrant object code.** Reentrancy allows many users to share the same code. Reentrancy is not an easy feature to achieve on the 370, especially if you use non-constant external variables, but we did it.
- **Optimization of the generated code.** We know the 370 instruction set and the various 370 operating environments. We have over 100 staff years of assembler language systems experience on our development team.
- **Generated code executable in both 24-bit and 31-bit addressing modes.** You can run compiled programs above the 16 megabyte line in MVS/XA.
- **Generated code identical for OS and CMS operating systems.** You can move modules between MVS and CMS without even recompiling.
- **Complete libraries.** We have implemented all the library routines described by Kernighan and Ritchie (the informal C standard), and all the library routines supported by Lattice (except operating system dependent routines), plus extensions for dealing with 370 operating environments directly. Especially significant is our byte-addressable Unix®-style I/O access method.
- **Built-in functions.** Many of the traditional string handling functions are available as built-in functions, generating in-line machine code rather than function calls. Your call to move a string can result in just one MVC instruction rather than a function call and a loop.

In addition to mainframe software development, you can also use our new cross-compiler to develop PC software on your IBM mainframe. With our cross-compiler, you can compile Lattice C programs on your mainframe and generate object code ready to download to your PC.

With the cross-compiler, we also offer PLINK86™ and PLIB86™ by Phoenix Software Associates Ltd. The Phoenix link-editor and library management facility can bind several compiled programs on the mainframe and download immediately executable modules to your PC.

Tomorrow...
We believe that the C language offers the SAS System the path to true portability and maintainability. And we believe that other companies will make similar strategic decisions about C. Already, C is taught in most college computer science curriculums, and is replacing older languages in many. And almost every computer introduced to the market now has a C compiler.

C, the language of choice...
C supports structured programming with superior control features for conditionals, iteration, and case selection. C is good for data structures, with its elegant implementation of structures and pointers. C is conducive to portable coding. It is simple to adjust for the size differences of data elements on different machines.

Continuous support...
At SAS Institute, we support all our products. You license them annually; we support them continuously. You get updates at no additional charge. We have a continuing commitment to make our compiler better and better. We have the ultimate incentive—all our software products depend on it.

For more information...
Complete and mail the coupon today. Because we've got the development tool for your tomorrow.

SAS Institute Inc.
SAS Circle, Box 8000
Cary, NC 27511-8000
Telephone (919) 467-8000 x 7000

I want to learn more about:

- ☐ the C compiler for MVS software developers
- ☐ the C compiler for CMS software developers
- ☐ the cross-compiler with PLINK86 and PLIB86

today...so I'll be ready for tomorrow.

Please complete or attach your business card.

Name ____________________

Title ____________________

Company ____________________

Address ____________________

City ____________ State ________ ZIP ________

Telephone ____________________

Mail to: SAS Institute Inc., Attn: CC, SAS Circle, Box 8000, Cary, NC, USA. 27511-8000. Telephone (919) 467-8000, x 7000 DDJ 8/87

Listing One

(Text begins on page 24.)

```
/*
 *
 *                          KROSS.C
 *
 *              COPYRIGHT (C) 1987 by Charles F. Bowman
 *
 *                      ALL RIGHTS RESERVED.
 *
 */
#include        "stdio.h"

#define NIL     '\000'

#define ALL     1
#define PUZ     2
#define DOWN    1
#define ACROSS  2

#define MINWORD 3
#define MAXPUZ  25
#define MAXWORD 50
#define WORDLEN 15

#define EMPTY   0
#define FREE    1
#define USED    2
#define SOLVED  3

#define BLANK   ' '
#define PADCHAR '-'
#define WORDS   "@words"
#define PUZZLE  "@puzzle"

#define FLAG(x, y)      list[ x - MINWORD ].w[ y ].flg
#define WORD(x, y)      list[ x - MINWORD ].w[ y ].word

FILE    *fp;
int     length, width;
char    puzzle[ MAXPUZ ][ MAXPUZ ];

struct  words   {
        char    word[ WORDLEN ];
        int     flg;
};

struct  {
        struct  words   w[ MAXWORD ];
} list[ WORDLEN - MINWORD ];

main( ac, av )
int     ac;
char    *av[];
{

        if( ac != 2 ){
                fprintf( stderr, "usage: kross puzzlefile\n" );
                exit( 1 );
        }
        if( (fp = fopen( av[1], "r" )) == NULL ){
                fprintf( stderr, "Cannot open'%s' to read!\n", av[1] );
                exit( 2 );
        }

        readpuz( fp );
        if( solve(0, -1) ){
                pprint( PUZ );
        } else {
                printf( "No Solution!!\n" );
        }
        exit( 0 );
}

/*
 *      ============================================================
 *              READPUZ(): read puzzle into memory from file
 *      ============================================================
```

(continued on page 52)

Listing One *(Listing continued, text begins on page 24.)*

```
  */
readpuz()
{
        int     i;
        char    buf[ 85 ];


        length = 0;
        /*
         *      Puzzle Section
         */
        if( fgets( buf, sizeof buf, fp ) == NULL ){
                fprintf( stderr, "%s: Premature EOF!\n", PUZZLE );
                exit( 4 );
        }
        if( strncmp( buf, PUZZLE, strlen( PUZZLE ) ) ){
                fprintf( stderr, "%s: BAD FORMAT!\n", PUZZLE );
                exit( 5 );
        }

        if(fgets(buf,sizeof buf,fp)==NULL
           || !strncmp(buf,WORDS,strlen(WORDS))){
                fprintf( stderr, "%s: Premature EOF!\n", PUZZLE );
                exit( 4 );
        }
        width = strlen( buf ) - 1;

        do {
                if( (strlen( buf ) - 1) != width ){
                        fprintf(stderr,"Line %d: badwidth!\n",width);
                        exit( 5 );
                }
                for( i = 0; i < width; i++ ){
                        if( buf[ i ] == BLANK ){
                                puzzle[ length ][ i ] = NIL;
                        } else if( buf[i] == PADCHAR ){
                                puzzle[ length ][ i ] = buf[ i ];
                        } else {
                                fprintf(stderr,"BAD CHAR'%d'L# %d\n",
                                        buf[i], length );
                                exit( 88 );
                        }
                }
                puzzle[ length ][ width ] = NIL;
                length += 1;
        } while(fgets(buf,sizeof buf,fp)!=NULL &&
                strncmp( WORDS, buf, strlen(WORDS) ) != 0 );

        /*
         *      Words Section
         */
        while( fgets( buf, sizeof buf, fp ) != NULL ){
```

(continued on page 54)

Listing One *(Listing continued, text begins on page 24.)*

```
		for( i = 0; i < MAXWORD; i++ ){
			if( FLAG( strlen(buf) - 1, i ) == EMPTY ){
				strncpy( WORD( strlen(buf) - 1, i ),
					buf, strlen(buf)-1 );
				FLAG( strlen(buf) - 1, i ) = FREE;
			}
		}
		if( i >= MAXWORD ){
			fprintf( stderr, "Out of space %d %s\n",
			    strlen(buf)-1, buf );
			exit( 6 );
		}
	}
	return;
}

/*
 *	==========================================================
 *		PPRINT(): display solved pzzle
 *	==========================================================
 */
pprint( t )
int	t;
{
	int	i, j;

	switch( t ){
	case ALL:
		/*
		 *	Debug only!
		 */
		for( i = MINWORD; i < WORDLEN; i++ ){
			j = 0;
			while( WORD(i, j)[0] != NIL ){
				printf( "%s\n", WORD(i, j) );
				j++;
			}
		}

	case PUZ:
		for( i = 0; i < length; i++ ){
			for( j = 0; j < width; j++ ){
				if( puzzle[ i ][ j ] ){
					putchar( puzzle[ i ][ j ] );
				} else {
					putchar( BLANK );
				}
			}
			putchar( '\n' );
		}
	}

	return;
}

/*
 *	==========================================================
 *		SOLVE(): function that searches for a solution
 *	==========================================================
 */
static	int	s = 0;
static	int	prev = -1;

solve( length, width )
int	length, width;
{
	int	l, w, i, len, tmp, type;
	char	old[ WORDLEN - MINWORD + 1 ];

	w = width;
	l = length;
	len = next( &l, &w, &type );
	if( len == 0 )
		return( SOLVED );

	for( i = 0; i < MAXWORD && WORD(len, i)[0] != NIL; i++ ){
		if( FLAG(len, i) == FREE
```

(continued on page 58)

Clarify and document your source listing and get an "organization chart" of your program's structure

with two NEW utilities from Aldebaran Laboratories, for C, BASIC, Pascal, dBASE,® FORTRAN and Modula-2 programmers.

Now works with FORTRAN

"Occasionally, a utility comes along that makes a programmer's life much easier. SOURCE PRINT is such a program. It contributes to the programmer's job by organizing code into a legible format and by helping to organize the documentation and debugging process."

— PC Magazine Sept. 16, 1986

Source Print and Tree Diagrammer both have easy-to-use menus with point-and-shoot file selection, and let you search for files containing a given string. For IBM PC and compatibles with 256K.

Join thousands of programmers who are working more efficiently using Source Print and Tree Diagrammer. Order these indispensable tools today. We ship immediately, and there's no risk with our 60-day money-back guarantee. **Order both and save. Only $155.00.**

800-257-5773 Dept. 58

In California:

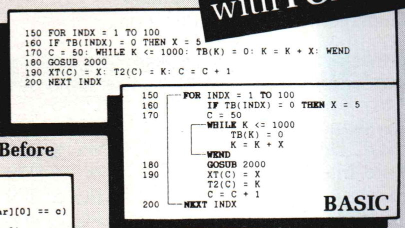
800-257-5774 Dept. 58

MasterCard, VISA, American Express, COD. Add $5 for shipping/handling.

or see your local dealer!

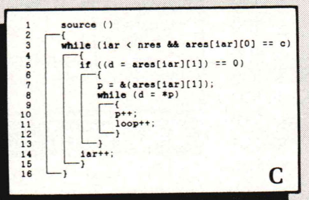
Source Print and Tree Diagrammer are trademarks of Aldebaran Labs. dBASE is a trademark of Ashton Tate. Prices subject to change without notice.

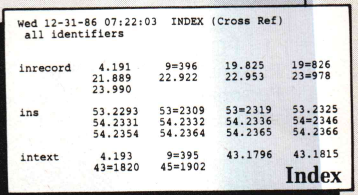
Source Print™



Before

After

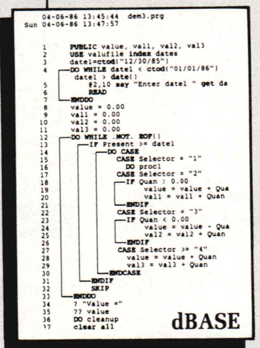




organizes your source code, simplifies debugging, and makes documentation a snap! It lists one or more source files with informative page headings and optional line numbers, while offering invaluable features:

The Index (Cross-Reference list) saves you time by showing exactly where variables are used and where functions, procedures, and routines are called. Locations where new values may be assigned to variables are shown, making it easy to track down that mysterious value change.

$97.00

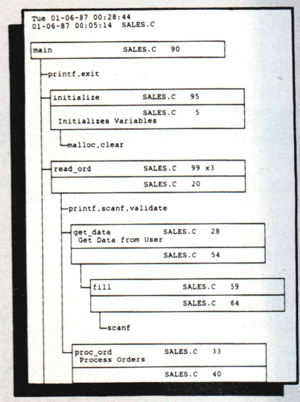


Structure Outlining solves the problem of hard-to-see nested control structures by automatically drawing lines around them.

Automatic Indentation of source code and listings reduces your editing time and ensures indentation accuracy.

Plus . . . Source Print generates a table of contents listing functions and procedures. Keywords can be printed in boldface on most printers. Multi-statement BASIC lines can be split for readability. Functions and procedures can be drawn by name from one or more source files to form a new file.

Tree Diagrammer™



shows your program's overall organization at a glance. Ordinary program listings merely display functions, procedures, and subroutines sequentially, but do not display the relationships between these routines. Our revolutionary new Tree Diagrammer automatically creates an "organization chart" of your program showing the hierarchy of calls to functions, procedures, and subroutines. Recursive calls are indicated and designated comments in the source code will appear on the chart.

Tree Diagrammer helps you organize your program more logically. And you'll be amazed at how easy it is to debug when you see how your routines interact.

$77.00

Aldebaran Laboratories 3339 Vincent Rd. Pleasant Hill, CA 94523 415-930-8966

YES! Rush me ☐ Source Print @ $97. ______ ☐ Tree Diagrammer @ $77. ______

☐ Both $155. Ship/Handling $5. For CA add 6% tax ______ Total ______

Name ______

Company ______

Address ______

City ______ State ______ Zip ______

☐ Check enclosed ☐ VISA ☐ MasterCard ☐ American Express

Card # ______ Exp. Date ______

Signature ______ Phone # ______

CIRCLE 350 ON READER SERVICE CARD

THE PROGRAMMER'S SHOP

provides complete information, advice, guarantees and every product for Microcomputer Programming.

— C Tools — SPECIAL PRICES

These tools at right represent 1000's of hours of development effort. They're also among the most stable and popular tools we sell. Choose the ones that fit your application, and give your productivity a boost. In doubt? Call one our C Specialists today.

Order before August 31, 1987, and <u>mention this ad</u> for these SPECIAL PRICES:

	List	Normal	SPECIAL
ctree by Faircom	$395	$315	**$279**
Greenleaf Function Library	$185	$139	**$109**
Communications Library	$185	$129	**$109**
Data Windows	$225	$159	**$139**
C Index + - stable, tested B + tree	$395	$319	**$289**
Halo by Media Cybernetics	$300	$209	**$189**

C Support-Systems

Advantage C + +	PC	$479
C Sharp - **realtime,** tasks.	PC	$495
C ToolSet - DIFF, xref, source	MS	$ 89
The HAMMER by OES Systems	PC	$129
Lattice Text Utilities	MS	$ 89
Multi-C - multitasking	PC	$135
PC LINT-Checker. Amiga $89	MS	$ 99
Pfantasy Pac - by Phoenix	PC	$619
PforCe + +	PC	$239
Pre-C - Lint-Like	MS	$155
Quickshell - script compiler	PC	$349
Sapiens V8 - 8M workspace	PC	$269
SECURITY LIB - Source $229	PC	$115
Timeslicer	PC	$265
with library source	PC	$895

C-Screens, Windows, Graphics

C Power Windows by Entelekon	PC	$109
Curses by Aspen Scientific	PC	$109
Curses by Lattice	PC	$ 89
dBASE Graphics for C	PC	$ 69
ESSENTIAL GRAPHICS - fast	PC	$185
GraphiC - new color version	PC	$285
Greenleaf Data Windows	PC	$159
w/source	PC	$289
LightWINDOWS/C-for Datalight C	PC	$
Multi-Windows - use w/ Multi-C	PC	$295
Screen Ace Form Master	PC	$195
Vitamin C - screen I/O	PC	$159
Windows for C - fast	PC	$189
Windows for Data - validation	PC	$319
View Manager - by Blaise	PC	$179
ZView - screen generator	MS	$169

Debuggers

386 Debug - by Phar Lap	PC	$129
Breakout - by Essential	PC	$ 89
CODESMITH - visual	PC	$ 99
C SPRITE - data structures	PC	$119
Periscope I	PC	$289
Periscope II	PC	$139
Periscope II-X	PC	$105
Pfix-86 Plus - by Phoenix	PC	$229
Turbo TDebug	PC	$ 55
Showcase - test software	PC	$125
SoftProbe II - embedded systems	PC	$695

FEATURE

C Scape II - by The Oakland Group. New version includes screen editor, still captures Dan Bricklin screens. Auto data array handling, horizontal scroll. PC $279

Fortran & Supporting

50:More FORTRAN	PC	$ 95
ACS Time Series	MS	$399
Forlib + by Alpha	MS	$ 59
I/O Pro - screen development	PC	$129
MACFortran by Microsoft	MAC	$229
MS Fortran - 4.0, full 77'	MS	$279
No Limit - Fortran Scientific	PC	$109
PC-Fortran Tools - xref, pprint	PC	$165
RM/Fortran	MS	Call
Scientific Subroutines - Matrix	MS	$129

Multilanguage Support

BTRIEVE ISAM	MS	$185
BTRIEVE/N-multiuser	MS	$455
Flash-Up Windows	PC	$ 79
GSS Graphics Dev't Toolkit	PC	$375
HALO Development Package	MS	$389
Informix 4GL-application builder	PC	$789
Informix SQL - ANSI standard	PC	$639
NET-TOOLS - NET-BIOS	PC	$129
Opt Tech Sort - sort, merge	MS	$ 99
PANEL	MS	$215
Pfinish - by Phoenix	MS	$229
PolyBoost - speed I/O, keyboard	PC	$ 69
Prime Factor FFT - 8087/287	PC	$145
PVCS Corporate-source control	MS	$309
PVCS Personal	MS	$109
QMake by Quilt Co.	MS	$ 79
Report Option - for Xtrieve	MS	$109
Screen Machine	PC	$ 59
Screen Sculptor	PC	$ 95
SRMS - new version	MS	$159
Synergy - create user interfaces	MS	$375
VXM - multi-env. link	MS	$195
Xtrieve - organize database	MS	$199
ZAP Communications - VT 100	PC	$ 89

Pascal and Supporting

ALICE - learn Pascal	PC	$ **49**
Exec - Chain Programs	MS	$ 79
MetaWINDOWS-graphics toolkit bit-mapped, fast	PC	$115
MetaWINDOWS PLUS	PC	$185
Microsoft PASCAL - faster	MS	$189
Pascal Pac with Tidy	PC	$ 49
Pascal Tools PLUS	PC	$119
Pascal 2 - by Oregon Software	MS	$329
Turbo Extender by Turbo Power	PC	$ 65
TurboHALO - 150 routines	PC	$ 99

DataBase & File Management

DataFlex by Data Access	PC	$ 899
Dataflex multiuser	PC	$1149
VP-Info - dBASE-like	PC	$ 79

RECENT DISCOVERY

C Worthy Interface Library - Complete, tested human interface for MS C, Lattice or Turbo C. Full screens, Windows, DOS, Error handling, Menus, Messages. Source separate, no royalties. PC $ 249

Other Languages

APL*PLUS/PC	PC	$ 429
CCS Mumps - Multiuser	PC	$ 359
Microsoft MASM	MS	$ 98
Modula-2 Apprentic Pkg.	PC	$ 79
Modula-2 Wizards Package	PC	$ 169
Pasm - by Phoenix	MS	$ 109
PC Forth + - by Lab Micro	PC	$ 199
Smalltalk/V	MS	$ 85
SNOBOL4 + - great for strings	MS	$ 80
UR/Forth	MS	$ 279

Xenix/Unix

Basic - by Microsoft	$ 209
C-Terp by Gimpel Software	$ 379
Cobol - by Microsoft	$ 609
Cobol Tools - by Microsoft	$ 319
Fortran or Pascal - by Microsoft	$ 419
Foxbase +	$ 689
MicroFocus Lev. II Compact COBOL	$ 795
Panel	$ 535
Real-Tools -	$ 89
RM/Cobol or RM/Fortran	Call
Xenix Complete System	$ 999
Xenix Development System	$ 499

Other Products

386 Assembler/Linker	PC	$ 389
Advantage Link	PC	$ 359
ASMLIB - 170 + routines	PC	$ 125
asmTREE - B + tree file mgmt.	PC	$ 339
Back-It - flexible, fast	PC	$ 89
Dan Bricklin's Demo Program	PC	$ 59
Disk Technician - smart disk upkeep	PC	$ 89
Help/Control - on line help	PC	$ 99
Interactive Easyflow-HavenTree	PC	$ 125
Link & Locate - Intel tools	MS	$ 329
LMK - like UNIX make	MS	$ 139
Microsoft Windows	PC	$ 69
Software Development Kit	PC	$ 319
MKS Toolkit - Unix, vi, awk	PC	$ 99
Norton Commander	PC	$ 55
Numerical Analyst by Magus	PC	$ 269
PLink - 86 PLUS - overlays	MS	$ 299
Polymake by Polytron	MS	$ 109
PolyShell by Polytron	MS	$ 109
PolyXREF by Polytron	PC	$ 99
PMaker - by Phoenix	PC	$ 79
Quelo 68000 X-ASM	PC	$ 509
Sapiens V8 - 8M virtual mgr.	PC	$ 269
Source Print - by Aldebaran	PC	$ 60
Taskview - ten tasks	PC	$ 55
Tree Diagrammer	PC	$ 45
Visible Computer: 8088	PC	$ 65
Xtree - classic graphic tree	PC	$ 45

Note: Mention this ad. Some prices are specials. Ask about COD and POs. Formats: 3" laptop now available, plus 200 others. UPS surface shipping add $3/item. All prices subject to change without notice.

Call for a catalog, literature, advice and service you can trust

VISA HOURS MasterCard

8:30 AM - 8:00 PM EST.

800-421-8006

THE PROGRAMMER'S SHOP ™

Your complete source for software, services and answers

5-L Pond Park Road, Hingham, MA 02043

Mass: 800-442-8070 or 617-740-2510 6/87

"I like your straightforward, open evaluations, comments and selection."

Chris Chapman
Practical Solutions Software

Listing One *(Listing continued, text begins on page 24.)*

```
                        && itfits(l, w, WORD(len, i), type) ){
                            FLAG(len, i) = USED;
                            enter( old, l, w, WORD(len, i), type );
                            prev = type;
                            tmp = solve( l, w );
                            if( tmp == SOLVED )
                                    return( SOLVED );
                            restore( old, l, w, type );
                            FLAG(len, i) = FREE;
                    }
            }

            return( 0 );
}

/*
 *      ==============================================================
 *              NEXT(): locate next slot to fill
 *      ==============================================================
 */
next( len, wht, t )
int     *len, *wht, *t;
{
        /*
         *      Return the next slot in the puzzle to attempt
         *      to be solved. DOWN has precedence.
         *
         *      The new values for len & wht will be updated.
         *      The returned value for the 'w' coordinate for
         *      an across 'hit' will have to be the value + 1.
         */
        int     l, w, tmp;

        l = *len;
        w = *wht;

        /*
         *      Check current position for across: down would
         *      have been done already.
         */
        if( w != -1 &&  ( (w - 1) < 0 || puzzle[l][w-1] == NIL )
        && puzzle[l][w] && (w + 1) < width && puzzle[l][w+1] ){
                /*
                 *      Across!
                 */
                *t = ACROSS;

                /*
                 *      Neccessary Evil!
                 */
                *wht = w + 1;

                tmp = 0;
                while( puzzle[l][w] != NIL && w < width ){
                        w += 1;
                        tmp += 1;
                }
                return( tmp );

        } else if( prev == DOWN || w == -1 ){
                w += 1;
        }

        /*
         *      Check for next possible position
         */
        for(; l < length; l += 1 ){
                for(; w < width; w += 1 ){
                        if( ( (l - 1) < 0 || puzzle[l-1][w] == NIL )
                        && puzzle[l][w] != NIL && (l + 1) < length
                        && puzzle[l+1][w] != NIL ){
                                /*
                                 *      Down!
                                 */
                                *t = DOWN;
```

(continued on page 60)

How to tackle a 300 page monster.

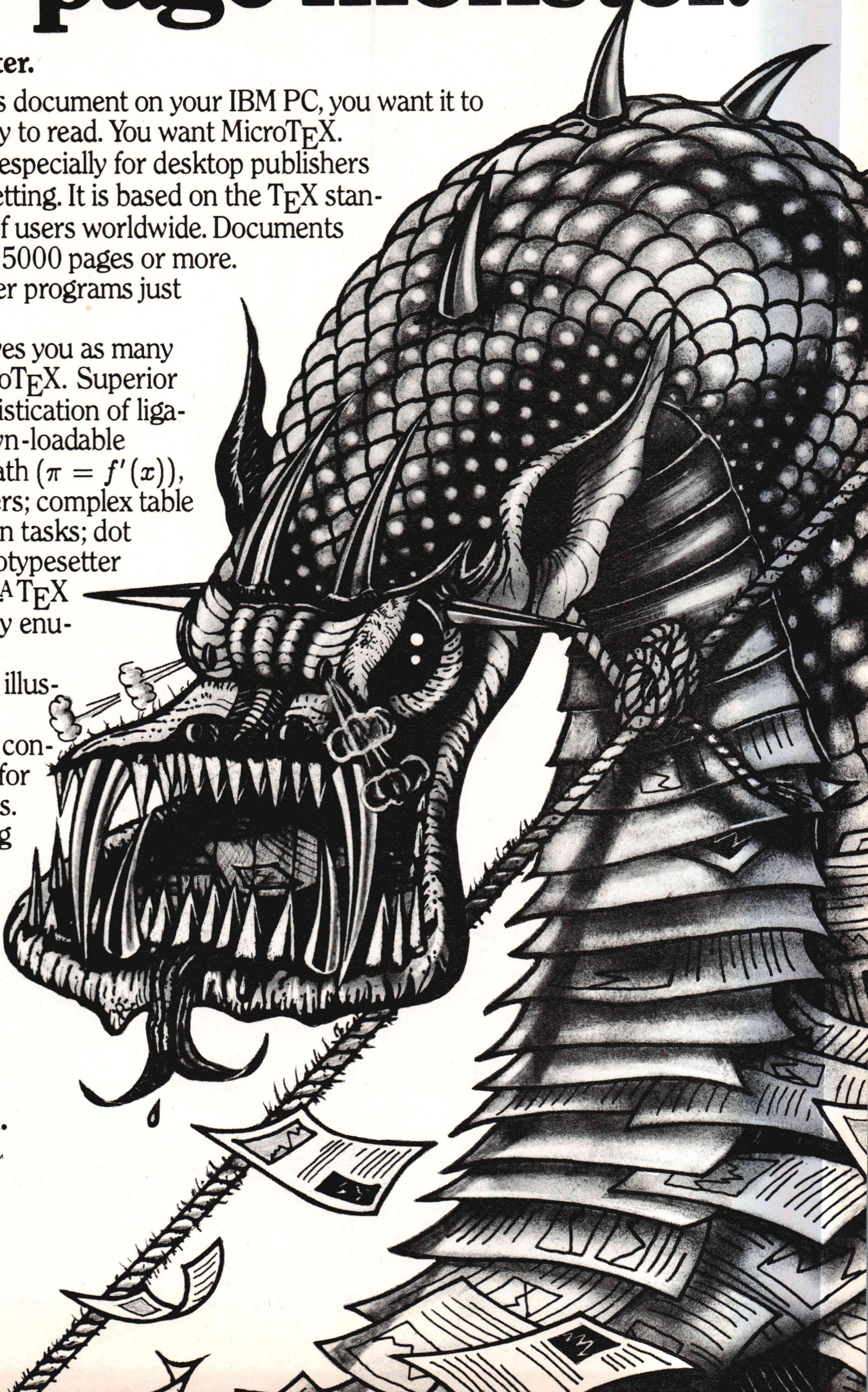
Turn your PC into a typesetter.

If you're writing a long, serious document on your IBM PC, you want it to look professional. Precise. Easy to read. You want MicroTeX.

MicroTeX was designed especially for desktop publishers who require heavy duty typesetting. It is based on the TeX standard, with tens of thousands of users worldwide. Documents from smaller than 30 pages to 5000 pages or more. And that's something that other programs just can't match.

No other PC software gives you as many advanced capabilities as MicroTeX. Superior hyphenation control, the sophistication of ligatures (ffi, fi) and kerning; down-loadable fonts; aesthetic handling of math ($\pi = f'(x)$), and foreign language characters; complex table construction and multi-column tasks; dot matrix, laser printer and phototypesetter output. When used with our LaTeX macro package, it automatically enumerates and cross-references pages, sections, footnotes and illustrations. Plus it automatically creates your indexes, tables of contents, and even updates them for you after last minute insertions.

So if you want typesetting software that's as serious as you are about your writing, get MicroTeX. ***Call*** 617-944-6795 to order or for more information.* Order with a 60-day money back guarantee.



Now available for Macintosh.

MicroTeX™
from Addison-Wesley

Serious typesetting for serious desktop publishers.

*Dealers, call our Dealer Hot Line: 800-447-2226 (In MA, 800-446-3399), ext. 2643.

CIRCLE 92 ON READER SERVICE CARD

Listing One *(Listing continued, text begins on page 24.)*

```
                                prev = DOWN;
                                *wht = w;
                                *len = 1;
                                tmp = 0;
                                while(puzzle[l][w]!=NIL && l<length){
                                        l += 1;
                                        tmp += 1;
                                }
                                return( tmp );
                        }
                        if( ((w - 1) < 0 || puzzle[l][w-1] == NIL)
                            && puzzle[l][w] && (w+1) < width
                            && puzzle[l][w+1] ){
                                /*
                                 *      Across!
                                 */
                                *t = ACROSS;
                                prev = ACROSS;
                                *len = 1;
                                *wht = w + 1;

                                tmp = 0;
                                if( w == -1 ) w = 0;
                                while(puzzle[l][w] != NIL &&w<width){
                                        w += 1;
                                        tmp += 1;
                                }
                                return( tmp );
                        }
                }
                w = 0;
        }

        /*
         *      Puzzle Completed!
         */
        return( 0 );
}

/*
 *      ==========================================================
 *              ITFITS(): determine is a word fits into a slot
 *      ==========================================================
 */
itfits( l, w, word, t )
char    *word;
int     t;
{
        char    *cp;

        if( t == ACROSS && w != -1)
                w -= 1;

        cp = word;
        while( *cp ){
                if( *cp != puzzle[l][w] && puzzle[l][w] != PADCHAR )
                        return( 0 );
                if( t == ACROSS )
                        w += 1;
                else
                        l += 1;
                cp++;
        }
        return( 1 );
}

/*
 *      ==========================================================
 *              ENTER(): enter word into puzzle
 *      ==========================================================
 */
enter( old, l, w, word, t )
char    *old;
int     l, w;
char    *word;
```

(continued on page 62)

HOT GRAPHICS PACKAGE FOR C PROGRAMS.* $39.95

Everything you need to write dramatic graphics effects into your Eco-C88 C programs. Some of the features include:

- Support for EGA, CGA, and Z100
- Over 100 graphics and support functions, many of which are PLOT-10 compatible.
- Many low level support routines reside outside your small model code-data area
- Can write dots thru the BIOS (for compatibility) or to memory (for speed)
- Graphics function help from CED editor available
- World, pixel or turtle color graphics modes
- 47 standard fill patterns, 17 line dashing patterns, Hershey fonts, plus user defineable fill, dash and fonts
- Supports view areas, rotateable fonts, clipping, arbitrary fill areas, extensive error checking, examples, and user's manual.

A must for the graphics enthusiast and a bargain at only

$39.95

***Requires Eco-C88 C Compiler.**

NEW POP-UP WINDOWS FOR YOUR C PROGRAMS.

This windowing library allows you to add pop up windows in your C programs quickly and easily. Use them for help windows, selection menus, error messages, special effects—anywhere you need an attention getter. Just some of the features include:

- CGA, EGA, and monochrome support
- Slow mode option for "flicker" displays
- Control any program that goes through the BIOS
- Use up to 255 windows
- No special window commands; use print f ()
- Resize and move windows
- Custom window titles and borders
- Can be used with ANSI device driver
- Most of window's code-data lies outside small model limits
- Use any of the IBM text or block characters
- User's manual and examples

The Windowing Library requires an IBM PC compatible BIOS and the Eco-C88 C compiler.

ONLY $29.95

HANDY LIBRARIAN MAKES LIFE EASIER.

Now you can combine your modules, functions, and subroutines into your own library for easy link commands. Fully compatible with ANY standard OBJ format files (not just Ecosoft's products). With the Ecosoft librarian, you can:

- Add, delete, and extract from a library
- Get table of contents or index of a library
- Combine libraries, control library page size, use switches for combinations, process complex library requests, use wildcards, and do library directives from command files.
- Complete with user's manual

A valuable addition for any programmer.

ONLY $29.95

THE FIRST PROFESSIONAL 'C' COMPILER FOR UNDER $60.

A C compiler with many ANSI enhancements at an unbelievably low price. The Eco-C88 C compiler has:

- Prototyping (the new type-checking enhancement)
- Enum and void data types
- Structure passing and assignment
- All operators and data types
- A standard library with more than 200 functions (many of which are System V compatible for greater code portability)
- CC and mini-make that all but automates the compile process
- 8087 support (we sense the 8087 at runtime — no dual libraries)
- ASM or OBJ output for use with MSDOS linker
- Tiered error messages — enable-disable lint-like error checking
- Fast compiles and executing code
- Expanded user's manual
- CED full-screen program editor

Everything you need at the unbelievable price of $59.95.

Eco-C88 C compiler requires an IBM PC, XT, or AT (or compatible) with 256K of memory, 2 disk drives and MSDOS 2.1 or later.

Orders only:
1-800-952-0472
Technical Information:
(317) 255-6476

NOT COPY PROTECTED.

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

ORDER FORM CLIP & MAIL TO: Ecosoft Inc., 6413 N. College Ave., Indianapolis, IN 46220

ITEM	PRICE	QTY	TOTAL
Flexi-Graph Graphics	$39.95		
Window Library	$29.95		
Eco-Lib Librarian	$29.95		
Eco-C88 C Compiler CED	$59.95		
		SHIPPING	
		TOTAL (IND. RES. ADD 5% TAX)	

PAYMENT: ☐ VISA ☐ MC ☐ AE ☐ CHECK

CARD # ______ EXPIR DATE ______

NAME ______

ADDRESS ______

CITY ______ STATE ______

ZIP ______ PHONE ______

CIRCLE 89 ON READER SERVICE CARD

ECOSOFT

Listing One *(Listing continued, text begins on page 24.)*

```
int         t;
{
            char    *cp;

            if( t == ACROSS )
                    w -= 1;

            cp = word;
            while( *cp ){
                    *old++ = puzzle[l][w];
                    puzzle[l][w] = *cp;
                    if( t == ACROSS )
                            w += 1;
                    else
                            l += 1;
                    cp++;
            }
            *old = NIL;

            return;
}

/*
 *          ==============================================================
 *                  RESTORE(): restore puzzle to prev state
 *          ==============================================================
 */
restore( old, l, w, t )
char        *old;
int         l, w, t;
{
            char    *cp;

            if( t == ACROSS )
                    w -= 1;

            cp = old;
            while( *cp ){
                    puzzle[l][w] = *cp;
                    if( t == ACROSS )
                            w += 1;
                    else
                            l += 1;
                    cp++;
            }

            return;
}
```

End Listing

Listing One *(Text begins on page 30.)*

```
/*---------------------------------------------------------------------------
-----------------------------------------------------------------------------
--           Name: DIFF.C
--      Processor: VAX | MS-DOS
--          Class: C Program
--  Creation Date: 1/8/87
--       Revision:
--         Author: D. Krantz
--
--    Description: File compare and change-bar for text files.
--
-----------------------------------------------------------------------------
---------------------------------------------------------------------------*/

/* File Difference Utility */

#include <ctype.h>
#include <stdio.h>

#define OPT_FLAG  '/'                  /* command line option switch recognizer   */

#ifdef VAX11C
#define MAXLINE 16                     /* maximum characters in input line        */
#else
#define MAXLINE 85
#endif

#define FORMFEED 'L'-'@'

struct LINE {                          /* structure defining a line internally    */
        int linenum;                   /* what line on page                       */
        int pagenum;                   /* what page line is from                  */
        struct LINE *link;             /* linked list pointer                     */
        char text[ MAXLINE ];          /* text of line                            */
        char dup[ MAXLINE ];           /* uppercase copy of line text             */
};

typedef struct LINE *line_ptr;

typedef char *char_ptr;

typedef FILE *FILE_PTR;

struct LINE root[ 3 ];                        /* root of internal linked lists    */

FILE_PTR msg;                                 /* differences summary file pointer */

int line_count[ 3 ] = { 1, 1, 1 };            /* file's line counter              */
int page_count[ 3 ] = { 1, 1, 1 };            /* file's page counter              */
int command_errors = 0;                       /* how many command line errors     */
char xx1[ 132 ], xx2[ 132 ];                  /* space to retain file names       */
int files = 0;                  /* how many files specified on command line       */
char_ptr infile_name[ 3 ] = { NULL, xx1, xx2 };
char outfile_name[ 132 ];        /* changebarred output filename                  */
FILE_PTR infile[ 3 ];                         /* input file pointers              */
FILE *outfile;                                /* changebarred output file pointer */
static line_ptr at[ 3 ] = { NULL, &(root[ 1 ]), &(root[ 2 ]) };

int debug = 0;                             /* trace switch                        */
int trace_enabled = 0;                /* keyboard tracing switch                  */
int bar_col = 78;                     /* column where change bar is to appear     */
int top_skip = 0;                             /* lines to skip at top of page     */
int bot_skip = 0;                             /* lines to skip at bottom of page  */
int page_len = 66;                            /* length of a page                 */
int up_case = 0;                     /* boolean, is upper/lower case significant? */
int re_sync = 5;             /* lines that must match for resynchronization       */
int output = 0;                      /* boolean, is change-barred output file on? */
int blanks = 0;                      /* boolean, are blank lines significant?     */
int lookahead = 200;         /* how many lines to look ahead before giving up     */
int skip1 = 0;                       /* how many pages of first file to skip      */
int skip2 = 0;                       /* how many pages of second file to skip     */

#if 0 /* tracing and other debug functions turned off */

#define trace( x )            callstack( x )
#define ret                   { callpop(); return; }
#define ret_val( x )          { callpop(); return( x ); }
#define TRACER_FUNCTIONS

#else
```

```
#define trace( x )          /** nothing **/
#define ret                 { return; }
#define ret_val( x )        { return( x ); }

#endif

/*---------------------------------------------------------------------------
---------------------------------------------------------------------------*/
main( argc, argv )
   int argc;
   char *argv[];
{
   int i;
         trace( "main" );
         if( argc == 1 )
                  help();
         msg = stdout;
         for( i = 1; i < argc; i++ )
                  strip_opt( argv[ i ] );
         if( files < 2 )
         {
                  printf( "\nError: Must specify two files" );
                  exit( 2 );
         }
         open_files();
         if( command_errors )
                  exit( 2 );
         page_skip();
         diff();
         ret;
}
/*---------------------------------------------------------------------------
DONT_LOOK - Tells us whether or not this line should be considered for
comparison or is a filler (e.g. header, blank) line.
---------------------------------------------------------------------------*/
dont_look( line )
   line_ptr line;
{
   int i;
         trace( "dont_look" );
         if( line == NULL )
                  ret_val( 0 );
         if( line->linenum <= top_skip )
                  ret_val( 1 );
         if( line->linenum > page_len - bot_skip )
                  ret_val( 1 );
         if( !blanks )
         {
                  for( i = 0; i < MAXLINE; i++ )
                           switch( line->text[ i ] )
                           {
                                    case '\0':
                                    case '\n':
                                             ret_val( 1 );
                                    case '\t':
                                    case ' ':
                                             break;
                                    default:
                                             ret_val( 0 );
                           }
         }
         ret_val( 0 );
}
/*---------------------------------------------------------------------------
EQUAL - tells us if the pointers 'a' and 'b' point to line buffers containing
equivalent text or not.
---------------------------------------------------------------------------*/
equal( a, b )
   line_ptr a, b;
{
         trace( "equal" );
         if( (a == NULL) || (b == NULL) )
                  ret_val( 0 );
         if( up_case )
                  ret_val( !strcmp( a->dup, b->dup ) )
         else
                  ret_val( !strcmp( a->text, b->text ) )
}
```

(continued on next page)

Listing One *(Listing continued, text begins on page 30.)*

```
/*---------------------------------------------------------------------------
POSITION - moves the input pointer for file 'f' such that the next line to
be read will be 'where'.
---------------------------------------------------------------------------*/
position( f, where )
    int f;
    line_ptr where;
{
        trace( "position" );
        at[ f ] = &root[ f ];
        while( at[ f ]->link != where )
                at[ f ] = at[ f ]->link;
        ret;
}

/*---------------------------------------------------------------------------
FIX - fixes the end-of-line sequence on a VAX to be just a newline instead of
a carriage-return/newline.
---------------------------------------------------------------------------*/
char *fix( str )
    char *str;
{
    char *strsave;

        trace( "fix" );
        strsave = str;
        if( str == NULL )
                ret_val( NULL )
#ifdef VAX11C
        while( *str != '\0' )
        {
                if( match( str, "\r\n" ) )
                {
                        *str = '\n';
                        *(str + 1) = '\0';
                }
                str++;
        }
#endif
        ret_val( strsave );
}

/*---------------------------------------------------------------------------
INDEX - returns a pointer to the first occurance of 'c' in the string pointed
to by 'str', or NULL if 'str' does not contain 'c'.
---------------------------------------------------------------------------*/
char *index( str, c )
    char *str, c;
{
        trace( "index" );
        while( (*str != c) && *(str++) );
        if( *str == c )
                        ret_val( str )
        ret_val( NULL );
}

/*---------------------------------------------------------------------------
NEXT_LINE - allocates, links, and returns the next line from file 'f' if no
lines are buffered, otherwise returns the next buffered line from file 'f'
and updates the link pointer to the next buffered line.
---------------------------------------------------------------------------*/
line_ptr next_line( f )
    int f;
{
    char *malloc();
    line_ptr temp, place_hold;

        trace( "next_line" );
        if( at[ f ]->link != NULL )
        {
                at[ f ] = at[ f ]->link;
                ret_val( at[ f ] );
        }
        else
        {
                at[ f ]->link = (line_ptr)malloc( sizeof( struct LINE ) );
                if( at[ f ]->link == NULL )
                {
                        printf( "\nOut of Memory" );
                        exit( 2 );
                }
```

(continued on page 70)

Listing One *(Listing continued, text begins on page 30.)*

```
                place_hold = at[ f ];
                at[ f ] = at[ f ]->link;
                at[ f ]->link = NULL;
                if( fix( fgets( at[ f ]->text, MAXLINE, infile[ f ] ) ) == NULL)
                {
                        free( at[ f ] );
                        at[ f ] = place_hold;
                        at[ f ]->link = NULL;
                        ret_val( NULL )
                }
#ifdef EMBEDDED_FORMFEEDS
                if( (index( at[ f ]->text, FORMFEED ) != NULL) ||
                        (line_count[ f ] > page_len ) )
#else
                if( ( *(at[ f ]->text) == FORMFEED) ||
                        (line_count[ f ] > page_len ) )
#endif
                {
                        page_count[ f ]++;
                        line_count[ f ] = 1;
                }
                at[ f ]->linenum = line_count[ f ]++;
                at[ f ]->pagenum = page_count[ f ];
                if( up_case )
                {
                        strcpy( at[ f ]->dup, at[ f ]->text );
                        upper( at[ f ]->dup );
                }
                ret_val( at[ f ] );
        }
}

/*-----------------------------------------------------------------------------
DISCARD - deallocates all buffered lines from the root up to and including
'to' for file 'f'.
-----------------------------------------------------------------------------*/
```

```
discard( f, to )
   int f;
   line_ptr to;
{
   line_ptr temp;

         trace( "discard" );
         for(;;)
         {
                  if( root[ f ].link == NULL )
                           break;
                  temp = root[ f ].link;
                  root[ f ].link = root[ f ].link->link;
                  free( temp );
                  if( temp == to )
                           break;
         }
         at[ f ] = &root[ f ];
         ret;
}

/*-------------------------------------------------------------------------
VFPUTS - for VAX, un-fixes newline at end of line to be carriage-return/newline.
-------------------------------------------------------------------------*/

vfputs( str, file )
   char *str;
   FILE *file;
{
   int i;
         trace( "vfputs" );
#ifdef VAX11C
         for( i = 0; i < MAXLINE; i++ )
         {
                  if( str[ i ] == '\n' )
                  {
                           strcpy( str + i, "\r\n" );
                           break;
                  }
         }
         fputs( str, file );
```

(continued on page 74)

WINDOWS FOR DATA™

The first choice of professional C programmers

"Windows for Data is the best programming tool I've ever used.
It's the most flexible I've seen.
Whenever I've wanted to do something,
I've been able to find a way."

Steven Weiss,
Stratford Systems

Professionals choose our tools because they are designed, crafted, and supported for professionals. Here at Vermont Creative Software, we understand that performance and pleasure in programming derive from more than a long list of functions. **Windows for Data** provides:

PROFESSIONAL FLEXIBILITY: Our customers repeatedly tell us how they've used WFD in ways we never imagined - but which we anticipated by designing WFD for unprecedented adaptability. Virtually every capability and feature can be modified to meet special needs. You will be amazed at what you can do with WFD.

PROFESSIONAL PERFORMANCE: Screen output is crisp and fast. Windows, menus, and data-entry forms snap up and down from the screen. WFD is built upon and includes **Windows for C, the windowing system rated #1 in speed and overall quality** in PC Tech Journal (William Hunt, July 1985).

PROFESSIONAL RELIABILITY: An unreliable tool is worse than no tool at all. VCS products are known in the industry for their exceptional reliability. Ask anyone who owns one.

PROFESSIONAL DOCUMENTATION: Over 600 pages of documentation provide step-by-step explanations for each major application, a reference page for each function, listings of functions alphabetically and by usage, and a fully cross-referenced index. Extensive tutorials and demonstration programs assist learning.

PROFESSIONAL TECHNICAL SUPPORT: The same expert programmers that develop our products provide prompt, knowledgeable technical support.

PROFESSIONAL PORTABILITY: High-performance versions of VCS products are available for XENIX, UNIX, and VMS, as well as DOS. No royalties on end-user applications.

OUR CHALLENGE AND GUARANTEE

If you have an application where no other tool can do the job, try **Windows for Data.** If it doesn't help you solve your problem, RETURN FOR A FULL REFUND. YOU MUST BE SATISFIED.

Ask for **FREE DEMO DISKETTE**

Vermont Creative Software

21 Elm Ave.
Richford, VT 05476
Telex: 510-601-4160 VCSOFT
Tel.: 802-848-7731

Prices: PCDOS* $395; XENIX, VMS, UNI
*PCDOS specify C compiler.

WINDOWS FOR DATA

for DOS, UNIX, VMS ...

The complete windowing data entry, menu, and help system that does the hard job others can't — we **guarantee** it!

Pop-up data entry windows; field types for all C data types, plus decimals, dates, and times; auto conversion to and from strings for all field types; system and user supplied validation functions; range checking; required, must-fill, and protected fields; free-form movement; multiple-choice field entry; scrollable sub-forms. Branch and nest windows, forms, and menus.

Complete context-sensitive help system with pop-up windows and scrollable text.

Pop-up, pull-down, scrollable, and Lotus-style menus.

NEW FOR DEBUGGING: Exclusive **VCS Error Traceback System** automatically identifies the location and cause of program errors. Eliminates the need to code error checks on all function calls! **VCS Memory Integrity Checking** helps catch those hard-to-detect, memory-corruption errors.

NEW FOR ERROR HANDLING: Install your own error handler to be called whenever a function detects an error.

NEW FORM LAYOUT UTILITY simplifies form design.

Listing One *(Listing continued, text begins on page 30.)*

```
#else
        fputs( str, file );
#endif
        ret;
}
/*----------------------------------------------------------------------------
PUT - If change-barred output file is turned on, prints all lines from the
root of file 1 up to and including 'line'. This is called only if a match
exists for each significant line in file 2.
----------------------------------------------------------------------------*/
put( line )
   line_ptr line;
{
   line_ptr temp;

        trace( "put" );
        if( output )
                for( temp = root[ 1 ].link; ; )
                {
                        if( temp == NULL )
                                ret
                        vfputs( temp->text, outfile );
                        if( temp == line )
                                ret
                        temp = temp->link;
                }
        ret;
}

/*----------------------------------------------------------------------------
CHANGE_BAR - inserts a change-bar into the text pointed to by
'str' and returns a pointer to 'str'.
----------------------------------------------------------------------------*/
char *change_bar( str )
   char *str;
{
   int i;
   char temp[ MAXLINE + 1 ], *dest,*base;

        trace( "change_bar" );
        base = str;
        dest = temp;
        i = 0;
        if( bar_col != 0 )
        {
                for( i = 0; *str != '\n'; i++ )
                {
                        if( (*str == '\r') && (*(str + 1) != '\n') )
                                i = 0;
                        *(dest++) = *(str++);
                }
                while( i++ < bar_col )
                        *(str)++ = ' ';
                strcpy( str, "|\n" );
        }
        else
                if( str[ 0 ] != ' ' )
                {
                        strcpy( temp, str );
                        strcpy( str + 1, temp );
                        str[ 0 ] = '|';
                }
        ret_val( base );
}

/*----------------------------------------------------------------------------
ADDED - Prints a change summary for all significant lines from the root of
file 1 up to and including 'line'. If output is enabled, adds a change bar
to the text and outputs the line to the output file.
----------------------------------------------------------------------------*/
added( line )
   line_ptr line;
{
   line_ptr temp;

        trace( "added" );
        for( temp = root[ 1 ].link; ; )
        {
                if( temp == NULL )
                        ret
                if( !dont_look( temp ) )
```

```
                        fprintf( msg, "+%d:%d -> %s", temp->pagenum,
                           temp->linenum, temp->text );
                if( output )
                        if( dont_look( temp ) )

                                vfputs( temp->text, outfile );
                        else
                                vfputs( change_bar( temp->text ),outfile );
                if( temp == line )
                        ret
                temp = temp->link;
        }
}

/*-------------------------------------------------------------------------
DELETED - outputs a change summary for all lines in file 2 from the root up to
and including 'line'.
-------------------------------------------------------------------------*/
deleted( line )
   line_ptr line;
{
   line_ptr temp;

        trace( "deleted" );
        for( temp = root[ 2 ].link; ; )
        {
                if( temp == NULL )
                        ret
                if( !dont_look( temp ) )
                        fprintf( msg, "-%d:%d -> %s", temp->pagenum,
                           temp->linenum, temp->text );
                if( temp == line )
                        ret
                temp = temp->link;
        }
        ret;
}

/*-------------------------------------------------------------------------
RESYNC - resynchronizes file 1 and file 2 after a difference is detected, and
outputs changed lines and change summaries via added() and deleted(). Exits
with the file inputs pointing at the next two lines that match, unless
it is impossible to sync up again, in which case all lines in file 1 are
printed via added(). Deallocates all lines printed by this function.
-------------------------------------------------------------------------*/
resync( first, second )
   line_ptr first, second;
{
   line_ptr file1_start, file2_start, last_bad1, last_bad2, t1, t2;
   int i, j ,k, moved1, moved2;

        trace( "resync" );

        moved1 = 0;
        file1_start = first;

        position( 1, first );
        for( k = 0; k < lookahead; k++ )
        {
                while( dont_look( file1_start = next_line( 1 ) ) );
                if( file1_start == NULL ) goto no_sy;

                moved2 = 0;
                file2_start = second;

                position( 2, second );
                for( j = 0; j < lookahead ; j++ )
                {
                        while( dont_look( file2_start = next_line( 2 ) ) );
                        if( file2_start == NULL ) goto eof2;

                        t1 = file1_start;
                        t2 = file2_start;
                        for( i = 0; (i < re_sync) && equal( t1, t2 ); i++ )
                        {
                                while( dont_look( t1 = next_line( 1 ) ) );
                                while( dont_look( t2 = next_line( 2 ) ) );
                                if( (t1 == NULL) || (t2 == NULL) )
                                        break;
                        }
                        if( i == re_sync ) goto synced;

                        last_bad2 = file2_start;
```

(continued on next page)

Listing One *(Listing continued, text begins on page 30.)*

```
                position( 2, file2_start );
                while( dont_look( file2_start = next_line( 2 ) ) );
                moved2 ++;

                }
eof2:
                last_bad1 = file1_start;
                position( 1, file1_start );
                while( dont_look( file1_start = next_line( 1 ) ) );
                moved1++;
        }
        printf( "\n*** ERROR - lost sync in file %s at page %d line %d",
                infile_name[ 1 ], first->pagenum, first->linenum );
        fclose( outfile );
        exit( 2 );
no_sy:
        position( 1, first );
        while( (first = next_line( 1 )) != NULL )
        {
                added( first );
                discard( 1, first );
        }
        ret;
synced:
        if( moved1 )
        {
                added( last_bad1 );
                discard( 1, last_bad1 );
        }
        position( 1, file1_start );
        if( moved2 )
        {
                deleted( last_bad2 );
                discard( 2, last_bad2 );
        }
        position( 2, file2_start );
        fprintf( msg, "\n" );
        ret;
}
```

```
/*----------------------------------------------------------------------------
DIFF - differencing executive. Prints and deallocates all lines up to where
a difference is detected, at which point resync() is called. Exits on end
of file 1.
----------------------------------------------------------------------------*/
diff()
{
   line_ptr first, second;

         trace( "diff" );
         for( ;; )
         {
                  while( dont_look( first = next_line( 1 ) ) );
                  if( first == NULL )
                  {
                           put( first );
                           ret;
                  }
                  while( dont_look( second = next_line( 2 ) ) );
                  if( equal( first, second ) )
                  {
                           put( first );
                           discard( 1, first );
                           discard( 2, second );
                  }
                  else
                           resync( first, second );
                  if( second == NULL )
                           ret
         }
}

/*----------------------------------------------------------------------------
PAGE_SKIP - skips the first 'skip1' pages of file 1, and then the first 'skip2'
pages of file 2. This is useful to jump over tables of contents, etc.
----------------------------------------------------------------------------*/
page_skip()
{
   line_ptr first, second;

         trace( "page_skip" );
         for( ; ; )
```

(continued on next page)

Listing One *(Listing continued, text begins on page 30.)*

```
        {
                first = next_line( 1 );
                if( (first == NULL) || (first->pagenum > skip1) )
                        break;
                put( first );
                discard( 1, first );
        }
        if( first != NULL )
                position( 1, first );
        for( ; ; )
        {
                second = next_line( 2 );
                if( (second == NULL) || (second->pagenum > skip2) )
                        break;
                discard( 2, second );
        }
        if( second != NULL )
                position( 2, second );
        ret;
}

/*---------------------------------------------------------------------------
HELP - outputs usage information.
---------------------------------------------------------------------------*/
help()
{
        printf( "\nDIFF" );
        printf( "\nText File Differencer and Change Barrer" );
        printf( "\n" );
        printf( "\nFormat:" );
        printf( "\nDIFF [option{option}] newfile oldfile [barfile]" );
        printf( "\n" );
        printf( "\n   newfile = latest revision of text file" );
        printf( "\n   oldfile = baseline to compare against" );
```

```
        printf( "\n   barfile = output file if changebars are desired" );
        printf( "\n" );
        printf( "\nOptions:" );
#ifdef TRACER_FUNCTIONS
        printf( "\n   /TRACE      Makes a mess of the display and runs real
                                                                  slow" );
        printf( "\n               default = trace off" );
        printf( "\n" );
#endif
        printf( "\n   /BAR_COL=n  Column of output file in which change bar
                                                 will appear" );
        printf( "\n               default = 78" );
        printf( "\n" );
        printf( "\n   /TOP_SKIP=n Lines at top of page to skip for running
                                             heads & page nos." );
        printf( "\n               default = 0" );
        printf( "\n" );
        printf( "\n   /BOT_SKIP=n Lines at botom of page to skip for running
                                            foots and page nos." );
        printf( "\n               default = 0" );
        printf( "\n" );
        printf( "\n   /PAGE_LEN=n Lines per page (embedded formfeeds over-
                                                          ride)" );
        printf( "\n               default = 66" );
        printf( "\n" );
        printf( "\n   /UP_CASE    Upper/Lower case is significant/is not
                                                 significant" );
        printf( "\n   /NOUP_CASE  default" );
        printf( "\n" );
        printf( "\n   /RE_SYNC=n  Lines that must match before files are
                                              considered synced" );
        printf( "\n               after differences are found - default = 5" );
        printf( "\n" );
        printf( "\n   /OUTPUT=file File to redirect differences summary to. " );
        printf( "\n               default = SYS$OUTPUT or console." );
        printf( "\n" );
        printf( "\n   /BLANKS     Blank lines are considered significant" );
```

(continued on page 81)

MICRO/ SYSTEMS JOURNAL

FOR THE ADVANCED COMPUTER USER

SUBSCRIBE NOW AND

SAVE OVER 15%

OFF THE NEWSSTAND PRICE!

15% SAVINGS

Yes! I want to subscribe to
Micro/Systems Journal™
and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______________________ Exp. date ____________

Signature ______________________________________

Name ______________________________________

Address ______________________________________

City ______________________ State ________ Zip ____________

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. **3019**

15% SAVINGS

Yes! I want to subscribe to
Micro/Systems Journal™
and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______________________ Exp. date ____________

Signature ______________________________________

Name ______________________________________

Address ______________________________________

City ______________________ State ________ Zip ____________

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. **3019**

15% SAVINGS

Yes! I want to subscribe to
Micro/Systems Journal™
and save over 15% off the cover price.

☐ 1 Year (6 issues) $20 ☐ 2 Years (12 issues) $35

☐ Please charge my: ☐ Visa ☐ MasterCard ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______________________ Exp. date ____________

Signature ______________________________________

Name ______________________________________

Address ______________________________________

City ______________________ State ________ Zip ____________

Savings based on a full one-year cover price of $23.70. Canada and Mexico add $3 for surface mail, $7 for airmail per year. All countries add $12 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. **3019**

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

For the Advanced Computer User
Micro/Systems Journal™
Box 3713
Escondido, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

For the Advanced Computer User
Micro/Systems Journal™
Box 3713
Escondidio, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

For the Advanced Computer User
Micro/Systems Journal™
Box 3713
Escondidio, CA 92025-9843

Listing One *(Listing continued, text begins on page 30.)*

```
        printf( "\n   /NOBLANKS    default" );
        printf( "\n" );
        printf( "\n   /LOOKAHEAD=n Lines to look ahead in each file to resync
                                         after difference" );
        printf( "\n                default = 200" );
        printf( "\n" );
        printf( "\n   /SKIP1=n     Pages in NEWFILE to skip before compare.
                                         Also sets /SKIP2" );
        printf( "\n                default = 0" );
        printf( "\n" );
        printf( "\n   /SKIP2=n     Pages in OLDFILE to skip before compare.
                                      Must be after /SKIP1" );
        printf( "\n                default = 0" );
        printf( "\n" );
}

/*------------------------------------------------------------------------
OPEN_FILES - opens the input and output files.
------------------------------------------------------------------------*/
open_files()
{
   int i;

        trace( "open_files" );
        for( i = 1; i < 3; i++ )
                if( (infile[ i ] = fopen( infile_name[ i ], "r")) == NULL )
                {
                        printf( "\nError: Can't open %s", infile_name[ i ]);
                        command_errors++;
                }
        if( files > 2 )
                if( (outfile = fopen( outfile_name, "w" )) == NULL )
                {
                        printf( "\nError: Can't create %s", outfile_name );
                        command_errors++;
                }
        ret;
}

/*------------------------------------------------------------------------
REDIRECT - performs output redirection under VAX 11 VMS.
------------------------------------------------------------------------*/
redirect( str )
   char *str;
{
   char filename[ 132 ], *ptr, *dest;

        trace( "redirect" );
        dest = filename;
        if( (ptr = index( str, '=' ) + 1) == (char *)(NULL + 1) )
        {
                printf( "\nERROR in option %s", str );
                command_errors++;
        }
        while( (*ptr != OPT_FLAG) && ((*(dest++) = *(ptr++)) != '\0') );
        *dest = '\0';
        if( (msg = fopen( filename, "w" )) == NULL )
        {
                printf( "\nERROR creating %s", filename );
                command_errors++;
        }
        ret;
}

/*------------------------------------------------------------------------
STRIP_OPT - processes each command line option.
------------------------------------------------------------------------*/
strip_opt( str )
   char *str;
{
        trace( "strip_opt" );
        upper( str );
        if( str[ 0 ] == OPT_FLAG )
        {
                if( match( str + 1, "BAR_COL" ) )
                        bar_col = num( str );
                else if( match( str + 1, "TOP_SKIP" ) )
                        top_skip = num( str );
                else if( match( str + 1, "BOT_SKIP" ) )
                        bot_skip = num( str );
```

(continued on next page)

WHAT'S THE DIFF?

Listing One *(Listing continued, text begins on page 30.)*

```
                else if( match( str + 1, "PAGE_LEN" ) )
                        page_len = num( str );
                else if( match( str + 1, "UP_CASE" ) )
                        up_case = 1;
                else if( match( str + 1, "NOUP_CASE" ) )
                        up_case = 0;
                else if( match( str + 1, "RE_SYNC" ) )
                        re_sync = num( str );
                else if( match( str + 1, "BLANKS" ) )
                        blanks = 1;
                else if( match( str + 1, "NOBLANKS" ) )
                        blanks = 0;
                else if( match( str + 1, "LOOKAHEAD" ) )
                        lookahead = num( str );
                else if( match( str + 1, "SKIP1" ) )
                        skip1 = skip2 = num( str );
                else if( match( str + 1, "SKIP2" ) )
                        skip2 = num( str );
#ifdef TRACER_FUNCTIONS
                else if( match( str + 1, "TRACE" ) )
                        trace_enabled = debug = 1;
#endif
                else if( match( str + 1, "OUTPUT" ) )
                        redirect( str );
                else
                {
                        printf( "\nUnrecognized Option: %s", str );
                        command_errors++;
                }
        }
        else
        {
                switch( files )
                {
                        case 0:
                                strcpy( infile_name[ 1 ], str );
                                break;
                        case 1:
                                strcpy( infile_name[ 2 ], str );
                                break;
                        case 2:
                                strcpy( outfile_name, str );
                                output = 1;
                                break;
                        default:
                                printf( "\nError: Too many files at %s",str);
                                command_errors++;
                                break;
                }
                files++;
        }
        if( index( str + 1, OPT_FLAG ) != NULL )
                strip_opt( index( str + 1, OPT_FLAG ) );
        ret;
}

/*---------------------------------------------------------------------------
UPPER - converts the string 'str' to upper case.
---------------------------------------------------------------------------*/
upper( str )
   char *str;
{
        trace( "upper" );
        for( ; ; )
        {
                if( *str == '\0' )
                        ret
                *str = toupper( *str );
                str++;
        }
}

/*---------------------------------------------------------------------------
MATCH - looks for a match of 'str' with the first (strlen( str) ) characters
of 'pattern'. Returns 0 for no match, nonzero on match.
---------------------------------------------------------------------------*/
int match( str, pattern )
   char *str, *pattern;
{
        trace( "match" );
        for( ; ; )
        {
```

(continued on page 84)

Listing One *(Listing continued, text begins on page 30.)*

```
                  if( *str != *pattern )
                           ret_val( 0 )
                  str++;
                  pattern++;
                  if( *pattern == '\0' )
                           ret_val( 1 )
                  if( *str == '\0' )
                           ret_val( 1 )
                  if( *str == '=' )
                           ret_val( 1 )
         }
}
/*------------------------------------------------------------------------------
NUM - returns the integer associated with a command line option. An equal
sign must appear in the option.
------------------------------------------------------------------------------*/
int num( str )
   char *str;
{
         trace( "num" );
         if( index( str, '=' ) == NULL )
                  ret_val( 0 )
         else
                  ret_val( atoi( index( str, '=' ) + 1 ) )
}

#ifdef TRACER_FUNCTIONS

char_ptr names[ 20 ];
int stack = 0;

callstack( str )
   char *str;
{
   int i;
   char c;

         names[ stack++ ] = str;
```

```
        if( debug )
        {
                for( i = 0; i < stack; i++ )
                        printf( "    " );
                printf( "Entering %s\n", str );
        }
#ifndef VAX11C
        if( trace_enabled && kbhit() )
        {
                switch( getch() )
                {
                        case 't':
                        case 'T':
                                debug = !debug;
                                break;
                        case 's':
                        case 'S':
                                printf( "\n-----------" );
                                for( i = stack - 1; i >= 0; i-- )
                                        printf( "\n%s", names[ i ] );
                                printf( "\n-----------\n" );
                                break;
                        default:
                                break;
                }
        }
#endif
}
callpop()
{
    int i;
        if( debug )
        {
                for( i = 0; i < stack; i++ )
                        printf( "    " );
                printf( "Exiting  %s\n", names[ stack ] );
        }
        stack--;
}
#endif
```

End Listing

Listing One *(Text begins on page 122.)*

```
Listing 1:  BASIC source code for the Sieve benchmark

1000 ' Sieve Benchmark Test
1001 ' Version 1.0  5/30/86  Namir C. Shammas
1010 DEFINT A-Z
1020 SIZE = 7000
1030 MAXITER = 10
1040 TRUE = 1: FALSE = 0
1050 DIM  FLAGS(SIZE)
1060 PRINT "START ";MAXITER;" ITERATION"
1065 TIME$ = "00:00:00.00"
1070 FOR ITER = 1 TO MAXITER
1080    COUNT = 0
1090    FOR I = 0 TO SIZE
1100      FLAGS(I) = TRUE
1110    NEXT I
1120    FOR I = 0 TO SIZE
1130      IF FLAGS(I) <> TRUE THEN 1210
1140        PRIME = I+I+3
1150        K = I+PRIME
1160        WHILE K <= SIZE
1170           FLAGS(K) = FALSE
1180           K = K + PRIME
1190        WEND
1200        COUNT = COUNT + 1
1210    NEXT I
1220 NEXT ITER
1225 PRINT "Time is ";TIME$
1230 PRINT COUNT;" PRIMES"
1240 END
```

End Listing One

Listing Two

```
Listing 2: Translated C source code for the Sieve benchmark

char *TIME_(), *balloc();
static int *FLAGS, count, false, i, iter, k, maxiter, prime, size, true;
static int it_1, it_2, it_3;
static char *st_1;
static int ml_1;
main(argc, argv)
    int argc;
    char *argv[];
    {
    bio_init(argc, argv, 1);
    /* Sieve Benchmark Test */
    /* Version 1.0  5/30/86  Namir C. Shammas */
    size = 7000;
    maxiter = 10;
    true = 1;
    false = 0;
    ml_1 = size+1;
    bfree(FLAGS);
    FLAGS = (int*)balloc((long)sizeof(int) * (size+1));
    BPRINT("s;i;s", "\006START ", maxiter, "\012 ITERATION");
    STIME_("\01300:00:00.00");
    it_1 = maxiter;

    for (iter = 1; iter <= it_1; ++iter)
{
   count = 0;
   it_2 = size;

   for (i = 0; i <= it_2; ++i)
   {
     FLAGS[i] = true;
   }
   it_3 = size;

   for (i = 0; i <= it_3; ++i)
   {
     if (FLAGS[i] != true)
       goto l_1210;
     prime = i + i + 3;
     k = i + prime;
```

```
      while (k <= size)
      {
         FLAGS[k] = false;
         k = k + prime;
      }
      count = count + 1;

      l_1210:;
     }
}
    BPRINT("s;s", "\010Time is ", TIME_(&st_1));
    BPRINT("i;s", count, "\007 PRIMES");
    bexit(0);
    bexit(0);
}
```

End Listing Two

Listing Three

Listing 3: BASIC source code for a root-seeking program

```
1010 DEFDBL A-H,P-Z : DEFINT I-O : CLS
1040 INPUT "Enter function number [1..3] ";N : PRINT
1050 IF (N < 1) OR (N > 3) THEN 1040
1060 INPUT "Enter guess ";X : PRINT : READ Accuracy, MAX.ITER
1070 DATA 1.0E-07, 50
1075 Iter = 0 : Diverge% = 1 : Diff = 2 * Accuracy
1080 WHILE ABS(Diff) > Accuracy ' Start root seeking method
1100   H = .01 : IF ABS(X) > 1 THEN H = H * X
1110   X2 = X     : GOSUB 1200 : F0 = F
1120   X2 = X + H : GOSUB 1200 : F1 = F
1130   X2 = X - H : GOSUB 1200 : F2 = F
1140   Diff = 2 * H * F0 /(F1 - F2) : X = X - Diff : Iter = Iter + 1
1170   IF (Iter > MAX.ITER) THEN Diverge% = 0
1180 WEND
1190 IF (Diverge% = 0) THEN Accuracy = 10 * Accuracy : GOTO 1075
1192 PRINT USING "Root = +#.#######^^^^";X : PRINT
1194 PRINT USING "Number of iterations = ##";Iter : PRINT
1196 PRINT USING "Accuracy = #.###^^^^";Accuracy : PRINT
1198 END
1200 'Subroutine to handle function catalogue
1210 ON N GOSUB 2100,2200,2300 : RETURN
2100 F = EXP(X2) - 3 * X2^2 : RETURN
2200 F = X2^2 - 5 * X2 + 6  : RETURN
2300 F = X2^3 - 5 * X2 + 10 : RETURN
```

End Listing Three

Listing Four

Listing 4: Translated C source code for a root-seeking program

```
typedef struct data
     {
     unsigned d_line;
     char *d_text;
     }DATA;
static DATA da_1[] = {1060, "1.0E-08, 50\n"};
double ABS(), EXP(), f_raise();
static int divergeI, iter, max_iter, n;
static double accuracy, diff, f, f0, f1, f2, h, x, x2;
DATA *data_stmts[] =
     {
     da_1, 0
     };
main(argc, argv)
     int argc;
     char *argv[];
     {
     bio_init(argc, argv, 1);
     CLS();

 l_1040:;
     INPUT("P ;i", "\035Enter function number [1..3] ", &n);
     BPRINT("");
      if (-((n < 1)) | -((n > 3)))
         goto l_1040;
     INPUT("P ;d", "\016Enter guess : ", &x);
     BPRINT("");
     BREAD(" d,i", &accuracy, &max_iter);
```

(continued on next page)

Listing Four *(Listing continued, text begins on page 122.)*

```
l_1075:;
    iter = 0;
    divergeI = 1;
    diff = 2 * accuracy;
    while (ABS(diff) > accuracy)
        {
        /* Start root seeking method */
        h = 0.01;
         if (ABS(x) > 1)
            h = h * x;
        x2 = x;
        pr_1200();
        f0 = f;
        x2 = x + h;
        pr_1200();
        f1 = f;
        x2 = x - h;
        pr_1200();
        f2 = f;
        diff = 2 * h * f0 / (f1 - f2);
        x = x - diff;
        iter = iter + 1;
         if (iter > max_iter)
            divergeI = 0;
        }
     if ((divergeI == 0))
        {
        accuracy = 10 * accuracy;
        goto l_1075;
        }
    UPRINT("\025Root = +#.#######^^^^", "d", x);
    BPRINT("");
    UPRINT("\032Number of iterations = ###", "i", iter);
    BPRINT("");
    UPRINT("\024Accuracy = #.###^^^^", "d", accuracy);
    BPRINT("");
    bexit(0);
    }
pr_1200()
    {
    /* Subroutine to handle function catalogue */
    switch (n)
        {
        case 1:
        pr_2100();
        break;
        case 2:
        pr_2200();
        break;
        case 3:
        pr_2300();
        break;

        }
    return;
 /* Subroutine number 1 */
        }

 pr_2100()
     {
     f = EXP(x2) - 3 * f_raise(x2, (double) 2);
     return;
 /* Subroutine # 2 */
        }

 pr_2200()
     {
     f = f_raise(x2, (double) 2) - 5 * x2 + 6;
     return;
 /* Subroutine # 3 */
        }

 pr_2300()
     {
```

```
f = f_raise(x2, (double) 3) - 5 * x2 + 10;
return;
bexit(0);
}
```

End Listing Four

Listing Five

```
Listing 5: BASIC source code for Find/Replace utility.

1000 ' Batch Find/Replace Utility  Version 1.1  2/7/86
1010 ' IBM PC                     BASICA version 2.0
1020 ' Copyright (c) 1987  Namir Clement Shammas
1030 '-----------------------------------------------------
1040 OPTION BASE 1
1050 DEFINT A-Z
1060 DIM FILENAME$(20),FIND.STR$(30)
1070 DIM REPLACE.STR$(30),REPLACE.FLAG(30),TEXT.LINE$(500)
1080 '
1090 TRUE  = 1 : FALSE = 0 'Set true/false
1100 MAX.LINES = 500 ' Current maximum number of lines read from a file
1110 MAX.STRINGS = 30 ' Number of find/replace strings
1120 MAX.FILES = 20 ' Maximum number of files
1140 CLS
1150 '
1160 T$ = "BATCH FILE FIND/REPLACE PROGRAM" : GOSUB 2290
1170 T$ = "VERSION 1.0" : GOSUB 2290 : PRINT : PRINT
1180 GOSUB 1560 'GET.FILENAMES : Get filenames
1190 GOSUB 1820 'GET.STRINGS : Get search/replace strings
1200 FOR K = 1 TO NUM.FILES
1210   GOSUB 2060 ' READ.LINES: Read text lines from a file
1220   CHANGED = FALSE
1230   FOR I = 1 TO NUM.STRINGS
1240     FOUND = FALSE
1250     FOR J = 1 TO NUM.LINES
1260         PTR = INSTR(TEXT.LINE$(J),FIND.STR$(I))
1270         WHILE PTR > 0
1280            IF (FOUND = TRUE) THEN 1330
1290               FOUND = TRUE
1300               LPRINT
1310               LPRINT "KEYWORD : ";FIND.STR$(I)
1330            LPRINT J;":";TEXT.LINE$(J)
1340            IF (REPLACE.FLAG(I) = FALSE) THEN 1440
1350                 CHANGED = TRUE
1360                 FIRST$ = ""
1370                 IF PTR > 1 THEN FIRST$ = MID$(TEXT.LINE$(J),1,(PTR-1))
1380                 LAST$ = ""
1390                 IF (PTR+LEN(FIND.STR$(I))) <= LEN(TEXT.LINE$(J))
                                                     THEN 1420
1400                     LAST$ = MID$(TEXT.LINE$(J),(PTR+LEN(FIND.STR$(I))))
1420                 TEXT.LINE$(J) = FIRST$ + REPLACE.STR$(I) + LAST$
1440             PTR = INSTR(PTR+1,TEXT.LINE$(J),FIND.STR$(I))
1450         WEND
1460     NEXT J
1470   NEXT I
1480 IF (CHANGED = TRUE) THEN GOSUB 2190 ' WRITE.LINES
1490 NEXT K
1500
1510 LPRINT CHR$(140) ' form feed
1520
1530 END
1540
1550 '-----------------------------------------------------
1560 ' GET.FILENAMES:  Subroutine to input filenames from the keyboard
1570 NUM.FILES = 0
1580 WHILE (NUM.FILES <= 0) OR (NUM.FILES > MAX.FILES)
1590     INPUT "Enter number of files ";NUM.FILES
1600     PRINT
1610 WEND
1620 FOR I = 1 TO NUM.FILES
1630 'REPEAT.LOOP1:
1640        PRINT "Enter filename # ";I;" ";
1650        INPUT FILENAME$(I) : PRINT
1660        ON ERROR GOTO 1750
1670        OPEN "I",1,FILENAME$(I)
1680        CLOSE #1
1690        ON ERROR GOTO 0
1700     IF FILENAME$(I) = "" THEN 1630
1710 NEXT I
1720 RETURN
1740 '-----------------------------------------------------
1750 'HANDLE: Error hander for bad filenames
1760 PRINT "File ";FILENAME$(I);" was not found"
1770 PRINT
1780 FILENAME$(I) = ""
1790 RESUME NEXT
1800
1810 '-----------------------------------------------------
1820 ' GET.STRINGS: Subroutines to input search/replace strings
1830 NUM.STRINGS = 0
1840 WHILE (NUM.STRINGS <= 0) OR (NUM.STRINGS > MAX.STRINGS)
1850     INPUT "Enter number of search/replace strings ";NUM.STRINGS
1860     PRINT
1870 WEND
1880 FOR I = 1 TO NUM.STRINGS
1890     REPLACE.STR$(I) = ""
1900     PRINT : PRINT "For string # ";I
```

(continued on next page)

Listing Five *(Listing continued, text begins on page 122.)*

```
1910      INPUT "    Enter string ";FIND.STR$(I)
1920      INPUT "    R)eplace F)ind ";A$ : PRINT
1930      IF (INSTR("Rr",MID$(A$,1,1)) = 0) THEN REPLACE.FLAG(I) =
                                 FALSE ELSE REPLACE.FLAG(I) = TRUE
1980      IF REPLACE.FLAG(I) = FALSE THEN 2020
1990           INPUT "Enter replacement string ";REPLACE.STR$(I)
2000           PRINT
2020 NEXT I
2030 RETURN
2040
2050 '----------------------------------------------------
2060 ' READ.LINES: Subroutines to read text lines
2070 LPRINT
2080 LPRINT "PROCESSING FILE : ";FILENAME$(K)
2090 OPEN "I",1,FILENAME$(K)
2100 NUM.LINES = 0
2110 WHILE (NOT EOF(1)) AND (NUM.LINES <= MAX.LINES)
2120    NUM.LINES = NUM.LINES + 1
2130    LINE INPUT #1,TEXT.LINE$(NUM.LINES)
2140 WEND
2150 CLOSE #1
2160 RETURN
2180 '----------------------------------------------------
2190 ' WRITE.LINES: Subroutines to write text lines
2200 OPEN "O",1,FILENAME$(K)
2210 FOR I = 1 TO NUM.LINES
2220   PRINT #1,TEXT.LINE$(I)
2230 NEXT I
2240 CLOSE #1
2250 RETURN
2270 '----------------------------------------------------
2280 ' Subroutine to center a message
2290 PRINT SPC(40 - LEN(T$)\2);T$
2300 RETURN
```

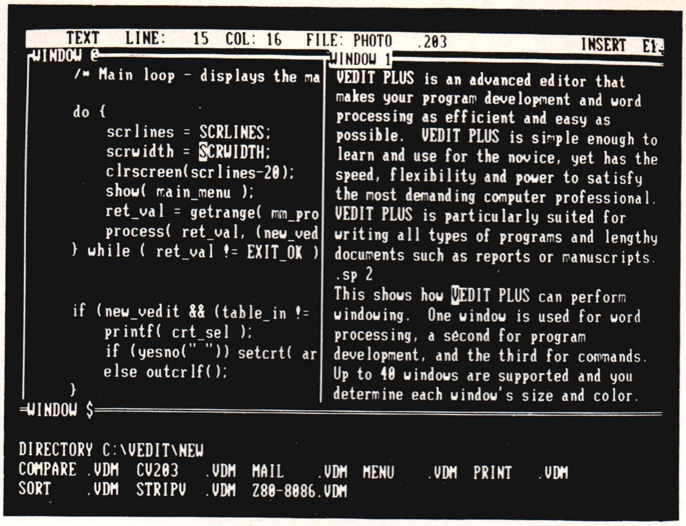
End Listing Five

Listing Six

```
Listing 6: Translated C source code for Find/Replace utility.

extern int on_error, err_code, err_stmt, trap_line, trap_err;
char *CHR_(), *MID_(), *s_asgn(), *s_cat();
int EOF(), INSTR(), LEN();
static int AREPLACE_FLAG[31], changed, false, found, i, j, k, max_files;
static int max_lines, max_strings, num_files, num_lines, num_strings, ptr;
static int true;
static char *FILENAME_[21], *FIND_STR_[31], *REPLACE_STR_[31];
static char *TEXT_LINE_[501], *a_, *first_, *last_, *t_;
static int it_1, it_2, it_3, it_4, it_5, it_6;
static char *st_1, *st_2;

main(argc, argv)
     int argc;
     char *argv[];
     {
     bio_init(argc, argv, 1);
     /* Batch Find/Replace Utility  Version 1.1  2/7/86 */
     /* IBM PC                   BASICA version 2.0 */
     /* Copyright (c) 1987  Namir Clement Shammas */
     /* ------------------------------------------------- */
     free_sp(FILENAME_, 21, 'S');
     free_sp(FIND_STR_, 31, 'S');
     free_sp(REPLACE_STR_, 31, 'S');
     free_sp(TEXT_LINE_, 501, 'S');

     true = 1;
     false = 0; /* Set true/false */
     max_lines = 500; /* Current maximum number of lines read from a file */
     max_strings = 30; /* Number of find/replace strings */
     max_files = 20; /* Maximum number of files */
     CLS();

     t_ = s_asgn(t_, "\037BATCH FILE FIND/REPLACE PROGRAM");
     sub_push(1);
     goto l_2290;
g_1:;
     t_ = s_asgn(t_, "\013VERSION 1.0");
     sub_push(2);
     goto l_2290;
g_2:;
E_0:;
     BPRINT("");
     if (err_code) {err_stmt=0; goto err_trap;}
E_1:;
     BPRINT("");
     if (err_code) {err_stmt=1; goto err_trap;}
E_2:;
     sub_push(3); /* GET.FILENAMES : Get filenames */
     goto l_1560;
g_3:;
     sub_push(4); /* GET.STRINGS : Get search/replace strings */
     goto l_1820;
g_4:;
     it_1 = num_files;
```

(continued on page 92)

286 / 386 'Protected Mode' Version Now Available



VEDIT PLUS

#1 PROGRAMMABLE EDITOR

FREE Fully Functional Demo Disk *

Stunning speed. Unmatched performance. Total flexibility. Simple and intuitive operation. The newest VEDIT PLUS defies comparison.

Try A Dazzling Demo Yourself.

The free demo disk is fully functional - you can try all features yourself. Best, the demo includes a dazzling menu-driven tutorial - you experiment in one window while another gives instructions.

The powerful 'macro' programming language helps you eliminate repetitive editing tasks. The impressive demo/tutorial is written entirely as a 'macro' - it shows that no other editor's 'macro' language even comes close.

Go ahead. Call for your free demo today. You'll see why VEDIT PLUS has been the #1 choice of programmers, writers and engineers since 1980.

Available for IBM PC, Tandy 2000, DEC Rainbow, MS-DOS, CP/M-86 and CP/M-80. (Yes! We support windows on most CRT terminals, including CRT's connected to an IBM PC.) Order direct or from your dealer. $185.

Compare features and speed

	BRIEF	Norton Editor	PMATE	VEDIT PLUS
'Off the cuff' macros	No	No	Yes	Yes
Built-in macros	Yes	No	Yes	Yes
Keystroke macros	Only 1	No	No	100+
Multiple file editing	20+	2	No	20+
Windows	20+	2	No	20+
Macro execution window	No	No	No	Yes
Trace & Breakpoint macros	No	No	Yes	Yes
Execute DOS commands	Yes	Yes	Yes	Yes
Configurable keyboard Layout	Hard	No	Hard	Easy
'Cut and paste' buffers	1	1	1	36
Undo line changes	Yes	No	No	Yes
Paragraph justification	No	No	No	Yes
On-line calculator	No	No	No	Yes
Manual size / index	250/No	42/No	469/Yes	380/Yes
Benchmarks in 120K File:				
2000 replacements	1:15 min	34 sec	1:07 min	6 sec
Pattern matching search	20 sec	Cannot	Cannot	2 sec
Pattern matching replace	2:40 min	Cannot	Cannot	11 sec

Call for 286 / XENIX Version
Fully Network Compatible

- Simultaneously edit up to 37 files of unlimited size.
- Split the screen into variable sized windows.
- 'Virtual' disk buffering simplifies editing of large files.
- Memory management supports up to 640K.
- Execute DOS commands or other programs.
- MS-DOS pathname support.
- Horizontal scrolling - edit long lines.
- Flexible 'cut and paste' with 36 'scratch-pad' buffers.
- Customization - determine your own keyboard layout, create your own editing functions, support any screen size.
- Optimized for IBM PC/XT/AT. Color windows. 43 line EGA.

EASY TO USE

- Interactive on-line help is user changeable and expandable.
- On-line integer calculator (also algebraic expressions).
- Single key search and global or selective replace.
- Pop-up menus for easy access to many editing functions.
- Keystroke macros speed editing, 'hot keys' for menu functions.

FOR PROGRAMMERS

- Automatic Indent/Undent for 'C', PL/I, PASCAL, etc.
- Match/check nested parentheses, i.e. '{' and '}' for 'C'.
- Automatic conversion to upper case for assembly language labels, opcodes, operands with comments unchanged.
- Optional 8080 to 8086 source code translator.

FOR WRITERS

- Word Wrap and paragraph formatting at adjustable margins.
- Right margin justification.
- Support foreign, graphic and special characters.
- Convert to/from WordStar and mainframe files.
- Print any portion of file; selectable printer margins.

MACRO PROGRAMMING LANGUAGE

- 'If-then-else', looping, testing, branching, user prompts, keyboard input, 17 bit algebraic expressions, variables.
- Flexible windowing - forms entry, select size, color, etc.
- Simplifies complex text processing, formatting, conversions and translations.
- Complete TECO capability.
- Free macros: • Full screen file compare/merge • Sort mailing lists • Print Formatter • Menu-driven tutorial

* Demo Disk is fully functional, but does not readily write large files.

CompuView

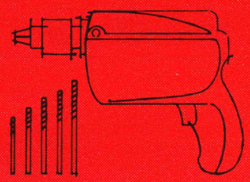
CIRCLE 122 ON READER SERVICE CARD

1955 Pauline Blvd., Ann Arbor, MI 48103 (313) 996-1299, TELEX 701821

Listing Six *(Listing continued, text begins on page 122.)*

```
    for (k = 1; k <= it_1; ++k)
        {
        sub_push(5); /* READ.LINES: Read text lines from a file */
        goto l_2060;
g_5:;
        changed = false;
        it_2 = num_strings;

        for (i = 1; i <= it_2; ++i)
            {
            found = false;
            it_3 = num_lines;
            for (j = 1; j <= it_3; ++j)
                {
E_3:;
                ptr = INSTR(-1, TEXT_LINE_[j], FIND_STR_[i]);
    if (err_code) {err_stmt=3; goto err_trap;}
E_4:;
                while (ptr > 0)
                    {
                     if ((found == true))
                        goto l_1330;
                    found = true;
E_5:;
                    BLPRINT("");
    if (err_code) {err_stmt=5; goto err_trap;}
E_6:;
                    BLPRINT("s;s", "\012KEYWORD : ", FIND_STR_[i]);
    if (err_code) {err_stmt=6; goto err_trap;}
E_7:;

l_1330:;
                    BLPRINT("i;s;s", j, "\001:", TEXT_LINE_[j]);
    if (err_code) {err_stmt=7; goto err_trap;}
E_8:;
                     if ((AREPLACE_FLAG[i] == false))
                        goto l_1440;
                    changed = true;
                    first_ = s_asgn(first_, "\000");
                     if (ptr > 1)
                        {
E_9:;
                        first_ = s_asgn(first_, MID_(&st_1, TEXT_LINE_[j],
                         1, (ptr - 1)));
    if (err_code) {err_stmt=9; goto err_trap;}
E_10:;
                        }
                    last_ = s_asgn(last_, "\000");
                     if ((ptr + LEN(FIND_STR_[i])) <= LEN(TEXT_LINE_[j]))
                        goto l_1420;
E_11:;
                    last_ = s_asgn(last_, MID_(&st_1, TEXT_LINE_[j], (ptr
                     + LEN(FIND_STR_[i])), -1));
    if (err_code) {err_stmt=11; goto err_trap;}
E_12:;

l_1420:;
                    TEXT_LINE_[j] = s_asgn(TEXT_LINE_[j], s_cat(&st_2, s
                                                              _cat(&st_1,
                     first_, REPLACE_STR_[i]), last_));

    if (err_code) {err_stmt=12; goto err_trap;}
E_13:;

l_1440:;
                    ptr = INSTR(ptr + 1, TEXT_LINE_[j], FIND_STR_[i]);
    if (err_code) {err_stmt=13; goto err_trap;}
E_14:;
                    }
                }
            }
         if ((changed == true))
            { /* WRITE.LINES */
            sub_push(6); /* WRITE.LINES */
            goto l_2190;
g_6:;
            }
        }
E_15:;
    BLPRINT("s", CHR_(&st_1, 140)); /* form feed */
    if (err_code) {err_stmt=15; goto err_trap;}
E_16:;
        bexit(0);
/* ------------------------------------------------- */

l_1560:;
    /* GET.FILENAMES: Subroutine to input filenames fromthe keyboard*/
    num_files = 0;
    while (-((num_files <= 0)) | -((num_files > max_files)))
        {
E_17:;
        INPUT("P ;i", "\026Enter number of files ", &num_files);
    if (err_code) {err_stmt=17; goto err_trap;}
```

(continued on page 94)



POWER TOOLS for SYSTEM BUILDERS™

TSF is owned and operated by programmers, so we understand your needs. We believe and practice integrity in all business dealings. We advertise our real prices and offer the best possible terms to all our customers. We provide prompt delivery and current versions.

We carry only products which we have personally used or which come with strong recommendations from programmers we respect. We have both the technical capability and the interest to meet your specific needs and to answer your technical questions. We gladly quote on special orders and volume discounts..

We accept checks, Visa, MasterCard, American Express and COD. We charge your card only when we ship and do not add a surcharge. We provide free UPS delivery on orders over $100. ($3 shipping on small orders. $2 COD fee.) We accept returns on most products. Give use a try!

The Software Family
649 Mission Street
San Francisco, CA 94105

800-443-7176

In California or outside U.S.

(415)583-4166

August Special: Blackstar "C" Functions only $99
Save over 20%

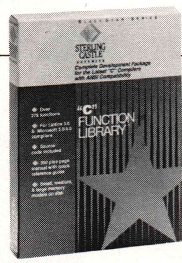
NEW! From Sterling Castle ...

BLACKSTAR "C" FUNCTIONS

Complete Development Package for the Latest "C" Compilers

Version 4.0 now includes:

- **Terminate and Stay Resident development**
- **EGA support**



BlackStar "C" Functions Library has over 300 functions. Supports the ANSI standard language and is compatible with the Microsoft version 3.0/4.0, Lattice version 3.0, and Borland Turbo C compilers.

Organized into easy to use modules, Blackstar "C" Function Library lets you concentrate on your user's specifications by providing pre-written solutions for the housekeeping functions required in almost every program. BlackStar "C" Function Library includes support for

- Files and Directories
- Date and Time
- Mouse Handler
- Windows
- Keyboard and Display
- Stay-Resident Programs
- Many more ...

The Blackstar "C" Library provides hundreds of tools to improve the quality of your programs and to improve your own programming productivity. Includes complete source code and supports both small and large memory models. The 375 page manual and four diskettes feature more explanations and additional demo programs. No royalties. At TSF's August special price of $99, its the best value in commercial quality C function libraries. Don't delay, call or write now to place your order! 30 day money back guarantee

STERLING CASTLE™
Sterling Castle, 702 Washington St., Suite 174, Marina del Rey, CA 90292

August Specials

Datalight Optimum C....... $95
Borland Turbo Basic $59
Borland Turbo C $59
BlackStar C Functions..... $99

Other TSF Values

Aldebaran **Source Print** ($75) **$59**
Blaise **C Tools +** (list $175) **$125**
Blaise **Light Tools** for Dataligh......... **$83**
Blaise **Turbo Power Tools +** **$79**
Borland **Turbo Basic** (list $99).... **$59**
Borland **Turbo C** (list $99) **$59**
Creative **Vitamin C** ($225) **$159**
Creative **VC Screen** ($100) **$79**
Datalight **Optimum C** w/Easy Edit **$95**
Gimpel **C-Terp** Specify compiler.... **$224**
Gimpel **PC-Lint** (list $139) **$103**
Greenleaf Data Windows **$157**
Guidelines **C++** (list $195)....... **$175**
Lattice **C** (list $500)..................... **$270**
Microsoft **C w/Codeview**.............. **$270**
Microsoft **Fortran w/Codeview** .. **$270**
Microsoft **Quick Basic** (list $99).. **$65**
MKS **Toolkit** Korn shell, vi, more **$115**
Periscope **Periscope I** ($345)...... **$279**
Periscope **Periscope II** ($175) **$139**
Periscope **Periscope III** ($995) .. **$799**
Phoenix **PFix-86+** (list $395) **$250**
Phoenix **Plink-86+** (list $495)... **$315**
Turbo Power **TDebug Plus** **$48**
Turbo Power **Optimizer** w/Source . **$99**

Call or write to request your FREE copy of our comprehensive catalog.

CIRCLE 230 ON READER SERVICE CARD

Listing Six

(Listing continued, text begins on page 122.)

```
E_18:;
        BPRINT("");
    if (err_code) {err_stmt=18; goto err_trap;}
E_19:;
        }
    it_4 = num_files;

    for (i = 1; i <= it_4; ++i)
        {

l_1630:;
        /* REPEAT.LOOP1: */
E_20:;
        BPRINT("s;i;s;", "\021Enter filename # ", i, "\001 ");
    if (err_code) {err_stmt=20; goto err_trap;}
E_21:;
        INPUT(" s", &FILENAME_[i]);
    if (err_code) {err_stmt=21; goto err_trap;}
E_22:;
        BPRINT("");
    if (err_code) {err_stmt=22; goto err_trap;}
E_23:;
        on_error = 1;
        err_code = 0;
        trap_line = 1;
E_24:;
        BOPEN("\001I", 1, FILENAME_[i], -1);
    if (err_code) {err_stmt=24; goto err_trap;}
E_25:;
        BCLOSE(1, 0);
        if (trap_err)
            {
            xer_msg(trap_err);
            bexit(1);
            }
        on_error = 0;
        err_code = 0;
         if (s_comp(FILENAME_[i], "\000") == 0)
            goto l_1630;
        }
    goto sub_ret;
/* ------------------------------------------------- */

l_1750:;
    /* HANDLE: Error hander for bad filenames */
E_26:;
    BPRINT("s;s;s","\005File ", FILENAME_[i],"\016 was not found");
    if (err_code) {err_stmt=26; goto err_trap;}
E_27:;
    BPRINT("");
    if (err_code) {err_stmt=27; goto err_trap;}
E_28:;
    FILENAME_[i] = s_asgn(FILENAME_[i], "\000");
    ++err_stmt;
    goto un_trap;
/* ------------------------------------------------- */

l_1820:;
    /* GET.STRINGS: Subroutines to input search/replace strings */
    num_strings = 0;

    while (-((num_strings <= 0)) | -((num_strings > max_strings)))
        {
E_29:;

        INPUT("P ;i", "\047Enter number of search/replace strings ",
                                                  &num_strings);
    if (err_code) {err_stmt=29; goto err_trap;}
E_30:;
        BPRINT("");
    if (err_code) {err_stmt=30; goto err_trap;}
E_31:;
        }
    it_5 = num_strings;

    for (i = 1; i <= it_5; ++i)
        {
        REPLACE_STR_[i] = s_asgn(REPLACE_STR_[i], "\000");
E_32:;
        BPRINT("");
    if (err_code) {err_stmt=32; goto err_trap;}
E_33:;
        BPRINT("s;i", "\015For string # ", i);
    if (err_code) {err_stmt=33; goto err_trap;}
E_34:;
        INPUT("P ;s", "\021    Enter string ", &FIND_STR_[i]);
    if (err_code) {err_stmt=34; goto err_trap;}
E_35:;
        INPUT("P ;s", "\023    R)eplace F)ind ", &a_);
    if (err_code) {err_stmt=35; goto err_trap;}
```

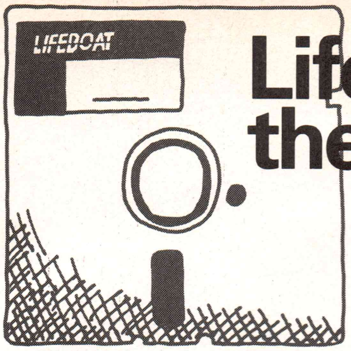
(continued on page 96)

Listing Six

(Listing continued, text begins on page 122.)


```
E_36:;
        BPRINT("");
    if (err_code) {err_stmt=36; goto err_trap;}
E_37:;
         if ((INSTR(-1, "\002Rr", MID_(&st_1, a_, 1, 1)) == 0))
            {
            AREPLACE_FLAG[i] = false;
            }
         else
            {
            AREPLACE_FLAG[i] = true;
            }
         if (AREPLACE_FLAG[i] == false)
            goto l_2020;
E_38:;
       INPUT("P ;s","\031Enter replacement string ", &REPLACE_STR_
                                                              [i]);
    if (err_code) {err_stmt=38; goto err_trap;}
E_39:;
       BPRINT("");
    if (err_code) {err_stmt=39; goto err_trap;}
E_40:;

l_2020:;
        }
    goto sub_ret;
/* ------------------------------------------------ */

l_2060:;
    /* READ.LINES: Subroutines to read text lines */
E_41:;
    BLPRINT("");
    if (err_code) {err_stmt=41; goto err_trap;}
E_42:;
    BLPRINT("s;s", "\022PROCESSING FILE : ", FILENAME_[k]);
    if (err_code) {err_stmt=42; goto err_trap;}
E_43:;
    BOPEN("\001I", 1, FILENAME_[k], -1);
    if (err_code) {err_stmt=43; goto err_trap;}
E_44:;
        num_lines = 0;
    while ((~(EOF(1))) & -((num_lines <= max_lines)))
        {
        num_lines = num_lines + 1;
E_45:;
        INPUT("FL1", 1, &TEXT_LINE_[num_lines]);
    if (err_code) {err_stmt=45; goto err_trap;}
E_46:;

        }
    BCLOSE(1, 0);
    goto sub_ret;
/* ----------------------------------------------- */

l_2190:;
    /* WRITE.LINES: Subroutines to write text lines */
E_47:;
    BOPEN("\001O", 1, FILENAME_[k], -1);
    if (err_code) {err_stmt=47; goto err_trap;}
E_48:;
    it_6 = num_lines;

    for (i = 1; i <= it_6; ++i)
        {
E_49:;
        BFPRINT(1, "s", TEXT_LINE_[i]);
    if (err_code) {err_stmt=49; goto err_trap;}
E_50:;  }
    BCLOSE(1, 0);
    goto sub_ret;
/* ------------------------------------------------- */
    /* Subroutine to center a message */
E_51:;

l_2290:;
    BPRINT("b;s", 40 - LEN(t_) / 2, t_);
    if (err_code) {err_stmt=51; goto err_trap;}
E_52:;
    goto sub_ret;
    bexit(0);

    sub_ret:
    switch(sub_pop())
        {
        case 1: goto g_1;
        case 2: goto g_2;
        case 3: goto g_3;
        case 4: goto g_4;
        case 5: goto g_5;
        case 6: goto g_6;
        }
```


(continued on page 98)



Lifeboat. Your best source for the best names in software.

C86PLUS

Proprietary C compiler design applies artificial intelligence to produce highly optimized code. Provides a highly productive and sophisticated programming environment, especially in ROM applications development. Features Microsoft C v. 4.0 and UNIX V compatibility; C library source code; full ANSI libraries and over 300 functions; ROMable code; small, compact, medium and large memory model support; 186/286/386 code generations options; and 8087/287 math support and auto detect emulator.

C86PLUS™

RUN/C Professional

Powerful C interpreter lets you dynamically load and unload compiled functions, execute in real-time at compiled speed, test modules with source code debugging and more. Includes more than 100 example programs; compatible with Lattice C and Microsoft C 4.0.

LIFEBOAT

Essential Comm Library Plus

A C library plus debugger stressing reliability and ease of use. It enables speeds to 9600 baud with XON/XOFF and XMODEM support. It includes a thorough manual with tutorial and easy-to-follow examples and demos. The debugger turns your PC into a sophisticated line monitor while an internal editor enables you to create, send or capture data, save it to a file, compute checksums or edit it in Hex or ASCII.

ADVANTAGE Make

Full featured. Supports UNIX MAKE scripts, multiple targets in single definition, full pathname, self-referencing macros. Many internally defined macros; i.e., $, $*, $< and much more.

Brand New LIFEBOAT

ADVANTAGE Disassembler

Provides immediate feedback as you work, storing results in tables on disk. Final output is ready for MS assembler. Handles .COM and .EXE files; 8086/186/286 code and 8087/287 coprocessors.

Brand New LIFEBOAT

PforCe++

Provides essential tools for object-oriented programming with C++. Speeds development time and ensures cleaner, more maintainable programs. You can build entire systems with a minimum of new coding. Over 400 easy-to-use functions include windowing, database, B-trees, field editing, menus, communication, table hashing/parsing, string/file handling and time/date calculations.

Phoenix

Windows for Data

Build a state-of-the-art user interface into your user program. Complete system for building and managing menus, data-entry forms, user help and text files in a windowing environment. Begins where others end. Features include: Field entry from lists of choices, scrollable regions for entry of a variable number of line items, nesting and branching of forms and menus. Unique built-in debugging system.

Vermont Creative Software

Periscope III

Periscope III sets a new standard in price/performance for real-time hardware breakpoint debuggers. You'll find the errors in real-time systems, stop intermittent failures, interface with undocumented systems and eliminate bottlenecks in your code. It's an easy transition from other models too, since the commands are a superset of those used in Periscope I, II and II-X. One board works on PC/XT/AT. The system includes the board, break-out switch, software, manual and quick reference card.

PERISCOPE

Windows Software Development Kit

Full-featured operating environment for developing and running application programs. Sophisticated user interface provides windowing, menus, dialog boxes and more. View and run multiple applications simultaneously; exchange data between programs; support keyboard control keys. Portable applications are independent of graphic devices. Full support for programs written in Microsoft C Compiler, Pascal and Macro Assembler. Includes utilities and sample code.

Microsoft®

Call today for your FREE Scientific & Engineering Software Guide!

Describes 100 of the best software packages for solving complex equations, number crunching, analyzing data and more.

Call **1-800-847-7078**
In NY **914-332-1875**
or see your local Lifeboat Affiliated Dealer

Call for the latest 80386 development software.

The Full-Service Source for Programming Software

LIFEBOAT

55 South Broadway
Tarrytown, NY 10591
Telex # 510-601-7602

The names of the products listed are generally the trademarks of the sources of the products.

INTERNATIONAL SALES OFFICES

Australia/New Zealand: MOS Computer Software/ Charlton Distributors Auckland (09) 766-361
Canada: Scantel Systems Toronto (416) 449-9252
Denmark: Ravenholm Copenhagen 288-7249
England: Grey Matter, Ltd. Devon 364-53499
System Science, Ltd. London (01) 248-0962
France: Compusol Paris 14 530 0737
Italy: Lifeboat Associates Italia Milan 02-464601
Japan: Lifeboat, Inc. Tokyo 03-293-4711
SATT Software Tokyo 03-295-3390
Netherlands: SCOS Automation BV Amsterdam 020-10 69 22
Spain: Micronet, S.A. Madrid 1-262-3304
Switzerland: Euro-Link Willisau 4145 813 514
West Germany: MEMA Computer GmbH Frankfurt 069-347226
Omnitex Rheinfelden 07623/61820

CIRCLE 245 ON READER SERVICE CARD

Listing Six

(Listing continued, text begins on page 122.)

```
err_trap: if (trap_err)
            {
            xer_msg(err_code);
            bexit(1);
            }
        trap_err = err_code;
        err_code = 0;
        goto l_1750;

un_trap: if (!trap_err)
            {
            xer_msg(-99);
            bexit(1);
            }
        trap_err = err_code = 0;
        switch(err_stmt)
            {
            case 0: goto E_0;
            case 1: goto E_1;
            case 2: goto E_2;
            case 3: goto E_3;
            case 4: goto E_4;
            case 5: goto E_5;
            case 6: goto E_6;
            case 7: goto E_7;
            case 8: goto E_8;
            case 9: goto E_9;
            case 10: goto E_10;
            case 11: goto E_11;
            case 12: goto E_12;
            case 13: goto E_13;
            case 14: goto E_14;
            case 15: goto E_15;
            case 16: goto E_16;
            case 17: goto E_17;
            case 18: goto E_18;
            case 19: goto E_19;
            case 20: goto E_20;
            case 21: goto E_21;
            case 22: goto E_22;
            case 23: goto E_23;
            case 24: goto E_24;
            case 25: goto E_25;
            case 26: goto E_26;
            case 27: goto E_27;
            case 28: goto E_28;
            case 29: goto E_29;
            case 30: goto E_30;
            case 31: goto E_31;
            case 32: goto E_32;
            case 33: goto E_33;
            case 34: goto E_34;
            case 35: goto E_35;
            case 36: goto E_36;
            case 37: goto E_37;
            case 38: goto E_38;
            case 39: goto E_39;
            case 40: goto E_40;
            case 41: goto E_41;
            case 42: goto E_42;
            case 43: goto E_43;
            case 44: goto E_44;
            case 45: goto E_45;
            case 46: goto E_46;
            case 47: goto E_47;
            case 48: goto E_48;
            case 49: goto E_49;
            case 50: goto E_50;
            case 51: goto E_51;
            case 52: goto E_52;
            }
    }
```

End Listings

PAINLESS WINDOWS.

Windows. Data Entry. Menus. Finally, a C programmers' tool that makes them as easy to use as *printf()*. With Greenleaf DataWindows™, you move in quantum leaps!

Snazzy Window Treatments

DataWindows represents an important breakthrough in C programming tools. It sets you free so you can create exciting programs quickly and easily, saving both time and money! Developed to work with the IBM PC, XT, AT, compatibles, and MSDOS or PCDOS, DataWindows is a carefully tooled system of C functions which will jazz up your programs with unprecedented efficiency.

Greenleaf DataWindows is integrated windows, transaction data entry, pop-up, pull-down, and Lotus style menu systems with:

- **Screen Management.** You don't have to remember what's on the display or the sequence in which you put it there. DataWindows does the grunt work. There are no restrictions.
- **Transaction Data Entry.** Data entry windows can have any number of fields with sophisticated options for reading many data types. Calls are made to help, validation, and other functions. Full featured text editing, protected and mandatory fields, dBASE type picture strings, context sensitive help, validation of fields and transactions, redefinable keys, password entry, attribute control, keyboard idle and much more.
- **Device Independence.** It detects the type of display adapter your computer is using and adjusts to it automatically for CGA, EGA, or monochrome. Logical video attributes are easy to use for color or monochrome.
- **Compatibility.** Runs with Microsoft Windows and IBM TopView.
- **The Greenleaf Tradition of Quality.** Reliable products. Professional documentation that gets you up and running quickly and keeps you there. Reference card. Newsletter and Bulletin board.

IBM, Microsoft & dBase, are registered trademarks of International Business Machines, Microsoft Corporation & Ashton-Tate respectively. PCDOS, IBM PC, XT, AT, & TopView are trademarks of IBM; MSDOS and Microsoft Windows are trademarks of Microsoft Corporation.

Stop Window Shopping

Order Today. Or call toll free for a free demo of the windows library that makes all the others obsolete.

Order any of these high performance tools by calling your dealer or 1-800-523-9830 today. Specify compiler when ordering. Add $8 for UPS second day air, or $5 for ground. Texas residents add sales tax. MasterCard, VISA, P.O., check, COD. In stock, shipped next day.

Greenleaf DataWindows	$225
DataWindows Source Module	$225
The Greenleaf Comm Library v2.0	$185
The Greenleaf Functions v3.0	$185
Digiboard Comm/4-II	$325
Digiboard Comm/8-II	$535

GREENLEAF
Software©

1411 LeMay Drive, Suite 101
Carrollton, TX 75007

Call Toll Free
1-800-523-9830
In Texas and Alaska, call
214-446-8641

CIRCLE 97 ON READER SERVICE CARD

Window Dressings

- **Simple or Complex Windows.** Up to 254 powerful overlaid windows simultaneously, all with just one kind of window to remember! Yet any window can be from one character to 32K!
- **Easy Window Operations.** DataWindows lets you move, zoom, frame, title, change colors, titles, frames, size, location, and make windows visible or invisible at will! Functions set cursor, attributes, and write data to any window or "current window". Word wrap, auto scroll, keyboard functions.
- **Write to Any Window Any Time.** Windows may be visible, overlaid, or invisible, and you can write to them anyway. What you write will be seen when the windows become visible.
- **DataWindows is fast!** It writes directly to video memory (in some modes).
- **Easy to save!** Any window, complete with attributes, can be saved on disk quickly and efficiently.
- **Source code available. No royalties.**

Also from Greenleaf:

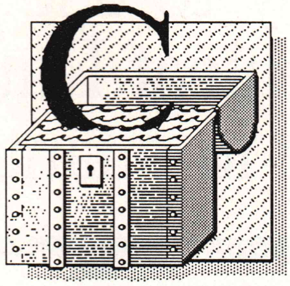
The Greenleaf Functions v3.0
The most complete, mature C language function library for the IBM PC, XT, AT and close compatibles. Includes over 225 functions — DOS, disk, video, color text and graphics, string, time/date, keyboard, disk status and Ctrl-Break functions plus many more.

The Greenleaf Comm Library
Our 2.0 version is the hottest communications facility of its kind. Over 120 functions — ring buffered, interrupt driven asynchronous communications for up to 16 ports simultaneously with XMODEM, XON/XOFF, many many sophisticated features.

We support all popular C compilers for MSDOS/PCDOS: Microsoft, Lattice, Computer Innovations, Aztec, DeSmet, and others.

Subroutines with A Variable Number of Arguments

by Allen Holub



I had intended this month to carry on with the adaptive Huffman tree stuff I started two months ago. After spending about 60 hours working on the code, I've finally given up. The paper I was working from didn't really provide enough information to implement the algorithm fully, and I got tired of having to decipher the thing. Maybe I'll pick up the project again at some future date, but for now I quit.

So, this month's C Chest is going to deal with a different topic entirely—subroutines with a variable number of arguments. I'm not going to present a specific program; rather, I'll discuss various techniques that you can use to write such subroutines and the sorts of problems you're likely to encounter. I'll discuss the ANSI and Unix methods for variable-argument passing at the end of the column. First, however, I'll look at what's actually going on.

Variable Arguments

At run time, all C subroutines use an area of the stack called a "stack frame" to hold arguments, local variables, and so forth. Here we're interested in the portion of the stack frame used for argument passing. Subroutine arguments are always passed on the stack, and the arguments are always pushed in reverse order (the rightmost argument is pushed first). So, for example, in *call (of, the, wild)* the variables *wild*, *the*, and *of* are pushed on the stack in that order. The number of bytes that are actually pushed depends on both the declared type of the variables and the automatic-type-conversion rules: both signed and unsigned variables of type *char* or *short int* are converted to *int* before being pushed. Similarly, variables of type *float* are converted to *double*. (This last automatic conversion is often compiler dependent, however, and can be suppressed with a function prototype). For the sake of the following examples, let's assume an 8-bit *char*, a 16-bit *int*, 32-bit *longs* and *floats*, and a 64-bit *double*.

Let's start with the following simple example; its stack is illustrated in Figure 1, below.

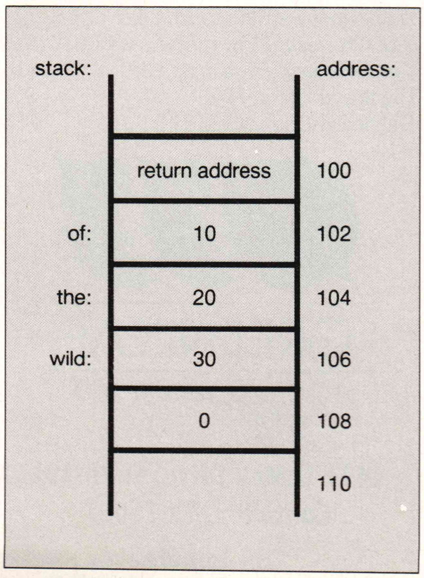


Figure 1: *Stack for the subroutine call* call(of, the, wild, 0);

```
int     of, the, wild;
void    call( int, . . . );

of      = 10;
the     = 20;
wild    = 30;
call( of, the, wild, 0 );
```

The function prototype says that *call( )* requires at least one *int*-size argument, followed by any number of additional arguments of indeterminate type.

The important thing to notice here is that, because all four arguments are of the same type and because they are at four contiguous memory locations, you can treat them as an array.

The *call( )* subroutine, shown in Example 1, page 102, prints any number of arguments, terminating when it sees a 0 argument. Only the first argument is declared because that's the only one you know will be present. The *argp* variable is a pointer to the additional arguments, which are treated as if they were an array of *ints*. *Argp* is initialized on line 6 to point at the first argument in the list—that is, *&argp* is a pointer to the *first* variable. It is of type *pointer-to-int*.

In this example, *&argp* is the number 102 (which is its address). From here on, you can treat the additional arguments as array elements, using the 0 to detect the last one. This same mechanism is used by several I/O library routines, such as *execl( )*, which takes a variable number of arguments all of the same type.

Of course, it's often necessary to pass arguments of different types. Consider the code in Example 2, page 103. The stacks resulting from the

calls to *nannuck( )*, on lines 35 and 36, are shown in Figures 2 and 3, below, where:

```
int      intvar;
float    floatvar;
```

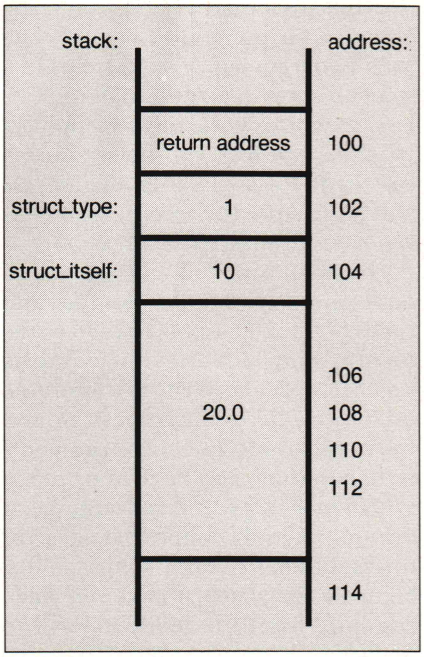


Figure 2: *Stack resulting from the call to* nannuck(1, 10, 20.0);

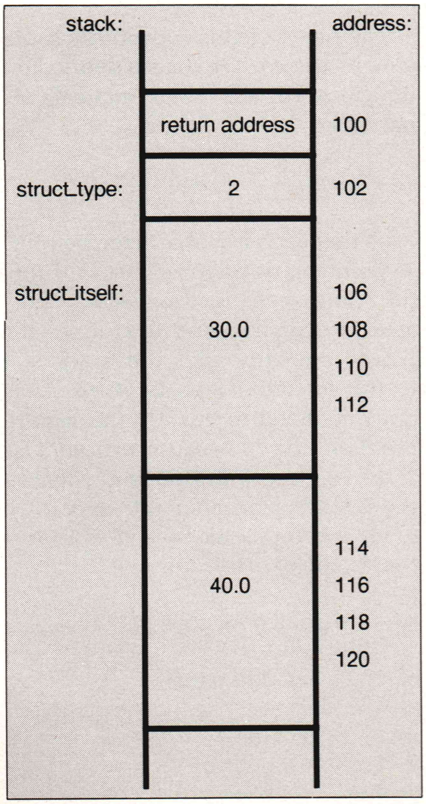


Figure 3: *Stack resulting from the call to* nannuck(2, 30.0, 40.0);

UNIX TOOLS FOR YOUR PC

(continued from page 101)

```
double   doublevar;
```

Here you're pushing the arguments on the stack in the normal way, but you're treating the resulting stack as a structure rather than as an array. The first argument is used to select which of the two possible structures is being passed. Note that you can't use a *char* or *short* in either structure because the corresponding argument will be converted to *int* as part of the subroutine call. By the same token, you can't use a *float* in the structure because all *floats* are converted to *doubles* as part of the call. Also note that this method won't necessarily be portable because you aren't guaranteed that the fields in a structure are contiguous. Nevertheless, it works with most compilers. Finally, note that you have to use a cast in the assignments on lines 21 and 27 because the second argument is not declared as a structure.

```
call( first )
int   first;
{
    int      *argp;

    argp = &first ;
    while( *argp != 0 )
            printf("%d\n", *argp++ );
}
```

Example 1: *The* call( ) *subroutine*

The main problem with the previous example is that the number and types of the arguments have to be determined in advance, at compile time. In order to write a subroutine such as *printf( )*, which doesn't know the number or types of its arguments until run time, you have to come up with a more sophisticated strategy. A *printf( )*-like subroutine that does just this is shown in Example 3, page 103. Figure 4, page 103, shows the stack resulting from the following call to *fang( )*:

```
fang("%c, %d, %s, %f0, '1', 2, "3", 4.5 );
```

The heart of this subroutine is obviously the *va_arg* macro defined on line 2. A *va_arg(argp, int)* call expands to:

```
((int *)(argp += sizeof(int)))[-1]
```

Note that *argp* is a character pointer, so pointer arithmetic is just arithmetic. That is, because the size of a character is 1, incrementing a character pointer actually adds the number 1 to the former contents. *Argp* starts out initialized to 102 (by the assignment on line 7). Because *sizeof(int)* is 2, the += sets it to 104. Now you cast the resulting number into a pointer to *int* and index backward from it—that is, the expression:

```
((int *)(argp += sizeof(int)))[-1]
```

can be treated like this:

```
int        *rvalue;

rvalue    = argp;
rvalue += sizeof( int );
rvalue [-1];
```

The code advances the pointer past the argument on the stack and then backs up (with the –1) to fetch the value. The cast to *int** forces the compiler to fetch an *int*-size argument from the stack. The += advances *argp* to point past this *int*-size argument. So, using this macro you can fetch any sort of argument from the stack, provided that you can tell the subroutine what the correct type is—information available to both *fang( )* and *printf( )* in the format string.

You could also break up the macro into two statements:

```
char *argp;
int    x;

x = * ((int *) argp);
argp += sizeof( int );
```

Here you cast *argp* into a pointer to

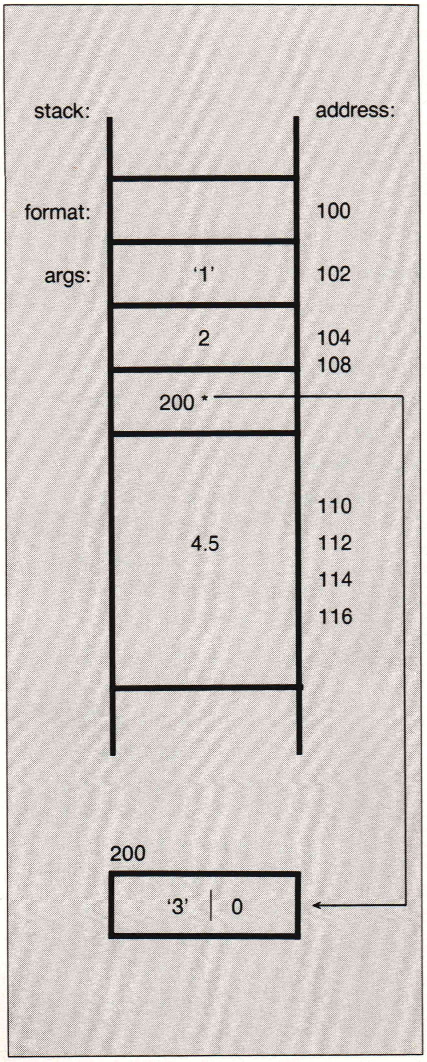


Figure 4: *The stack resulting from a* fang( ) *call*

```
struct the_first
{
    int      one;
    double   two;
};

struct the_second
{
    double one;
    double two;
};

nannuck( struct_type, struct_itself )
int     struct_type;
{
    struct the_first   *firstp;
    struct the_second *secondp;

    if( struct_type == 1 )
    {
        firstp = (struct the_first *) &struct_itself;
        printf("%d, %f\n", firstp->one,
                           firstp->two );
    }
    else
    {
        secondp = (struct the_second *) &struct_itself;
        printf("%f, %f\n", secondp->one,
                           secondp->two );
    }
}

main()
{
        nannuck(1,  10,    20.0 );
        nannuck(2,  30.0,  40.0 );
}
```

Example 2: *Using structures to access subroutine arguments*

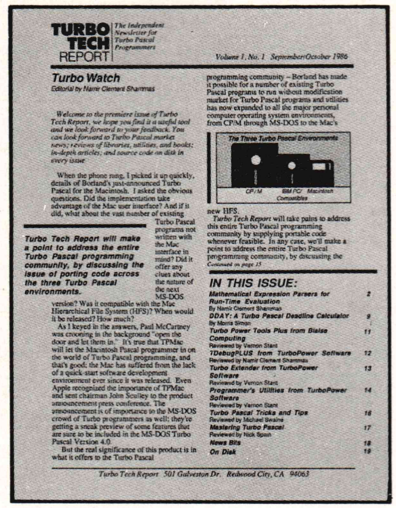
```
#include <stdio.h>
#define va_arg(argp,type)  ((type *)(argp += sizeof(type)))[-1]

fang( format, args )
char    *format;
{
    char        *argp = (char *) &args;

    for(;  *format ; format++ )
    {
        if( *format != '%' )
            putchar( *format );
        else
        {
            switch( *++format )
            {
            case 'c': printf("%c", va_arg(argp,int    ) ); break;
            case 'd': printf("%d", va_arg(argp,int    ) ); break;
            case 's': printf("%s", va_arg(argp,char *) ); break;
            case 'f': printf("%f", va_arg(argp,double) ); break;
            }
        }
    }
}

main()
{
        fang("%c, %d, %s, %f\n", '1', 2, "3", 4.5 );
}
```

Example 3: *A* printf( )*-like subroutine*

Turbo Tech Report Speaks Your Language.

Turbo Pascal Articles and Reviews

News and commentary



TURBO TECH REPORT — The Independent Newsletter for Turbo Pascal Programmers

Volume 1, No. 1 September/October 1986

Turbo Watch

Editorial by Namir Clement Shammas

Welcome to the premiere issue of Turbo Tech Report, we hope you find it a useful tool and we look forward to your feedback. You can look forward to Turbo Pascal market news; reviews of libraries, utilities, and books; in-depth articles; and source code on disk in every issue.

When the phone rang, I picked it up quickly, details of Borland's just-announced Turbo Pascal for the Macintosh. I asked the obvious questions. Did the implementation take advantage of the Mac user interface? And if it did, what about the vast number of existing Turbo Pascal programs not written with the Mac interface in mind? Did it offer any clues about the nature of the next MS-DOS version? Was it compatible with the Mac Hierarchical File System (HFS)? When would it be released? How much?

Turbo Tech Report will make a point to address the entire Turbo Pascal programming community, by discussing the issue of porting code across the three Turbo Pascal environments.

As I keyed in the answers, Paul McCartney was crooning in the background "open the door and let them in." It's true that TPMac will let the Macintosh Pascal programmer in on the world of Turbo Pascal programming, and that's good; the Mac has suffered from the lack of a quick-start software development environment ever since it was released. Even Apple recognized the importance of TPMac and sent chairman John Sculley to the product announcement press conference. The announcement is of importance to the MS-DOS crowd of Turbo programmers as well; they're getting a sneak preview of some features that are sure to be included in the MS-DOS Turbo Pascal Version 4.0.

But the real significance of this product is in what it offers to the Turbo Pascal programming community – Borland has made it possible for a number of existing Turbo Pascal programs to run without modification market for Turbo Pascal programs and utilities has now expanded to all the major personal computer operating system environments, from CP/M through MS-DOS to the Mac's new HFS.

The Three Turbo Pascal Environments

CP/M IBM PC/Compatibles Macintosh

Turbo Tech Report will take pains to address this entire Turbo Pascal programming community by supplying portable code whenever feasible. In any case, we'll make a point to address the entire Turbo Pascal programming community, by discussing the

Continued on page 15

IN THIS ISSUE:

Turbo Tech Report 501 Galveston Dr. Redwood City, CA 94063

A disk filled with Turbo Pascal code!

The newsletter/disk publication for Turbo Pascal® users

Are you looking for powerful utilities written in Turbo Pascal that you can use to develop software or incorporate into your programs? Are you interested in improving and expanding your Turbo Pascal programming skills?

Then you deserve a subscription to *Turbo Tech Report*, the bimonthly newsletter/disk publication from the publishers of *Dr. Dobb's Journal* and *Micro/Systems Journal*. Each issue delivers more than 250K of Turbo Pascal source code programs on disk, and 24+ pages of articles, Turbo Pascal software and book reviews, and analysis and commentary. It's the only publication delivering such focused technical articles with code on disk. Each valuable issue contains:

- ***Articles*** on topics like speedy 3D graphics, mathematical expression parsers, creating global gotos, memory resident and AI applications and more—all written by Turbo experts.
- ***Reviews*** of the latest Turbo Pascal software programs from companies like Borland International, Blaise Computing, Media Cybernetics, Nostradamus, and more!
- ***News and commentary*** detailing the latest products and developments in the Turbo Pascal programming community.
- ***A disk filled with Turbo Pascal code!*** You'll get the Turbo Pascal utilities and routines discussed in the newsletter's articles, as well as applications developed by Turbo users from around the world. You'll receive programs that make labels, generate menus, provide faster screen access, transfer files, and more!

If you're an expert Turbo Pascal programmer or a novice interested in expanding your Turbo skills, you need a publication that speaks your language: *Turbo Tech Report*. Subscribe today at the special price of just $99—that's 33% off the regular price of $150. To order by credit card, call toll-free 1-800-533-4372. Or mail the coupon with your payment to *Turbo Tech Report*, 501 Galveston Drive, Redwood City, CA 94063.

—Yes! I want a one-year subscription to *Turbo Tech Report* (6 issues with 6 disks) for $99.

Format: ☐ PC/MS-DOS ☐ Macintosh

CP/M: ☐ Kaypro ☐ Osborne ☐ Apple

PAYMENT MUST ACCOMPANY ALL ORDERS

☐ Check/money order enclosed.

☐ Charge my: ___ VISA ___ M/C ___ AmExp.

Card # ____________________ Exp. __________

Signature ____________________

Name ____________________

Address ____________________

City ______________ State ______ Zip ________

Orders outside U.S.: add $30.

CIRCLE 119 ON READER SERVICE CARD

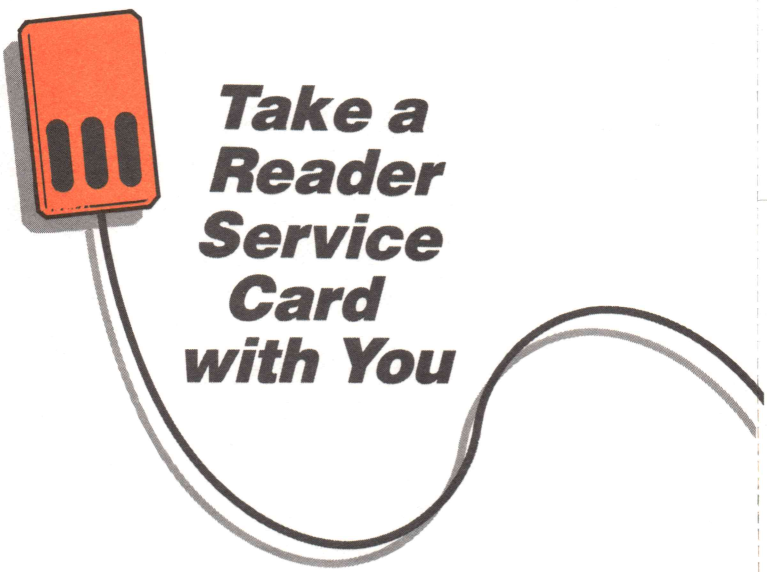
For Free Info ...

Start Here

Smart buyers start with *DDJ*'s free information card, a shopping center filled with information about the products and services advertised in this very issue: everything from software and systems to peripherals and professional support services.

And smart buyers can use this free information card to quickly and easily gather a comprehensive file of facts, figures and product specs to sort out competing claims. Using *DDJ*'s free information card can prevent you from making the wrong, costly buying decision.

Be a smart shopper. Complete and mail this postage paid card today!



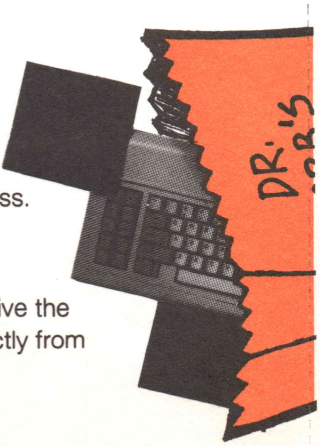
Take a Reader Service Card with You

It's Easy as ...

1. Circle the appropriate free information numbers, referring to the advertiser index for more information.

2. Fill in your name and address.

3. Mail today—postage is absolutely free. You'll receive the product information you need directly from the manufacturers, thanks to *DDJ*.



NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER
P.O. Box 2157
Clinton, Iowa 52735-2157

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #217, CLINTON, IOWA

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER
P.O. Box 2157
Clinton, Iowa 52735-2157

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

August '87: Use before November 30, 1987

Name ______________________________

Title ______________________________

Company ____________________ Phone ____________

Address ______________________________

City/State/Zip ______________________________

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

August '87: Use before November 30, 1987

Name ______________________________

Title ______________________________

Company ____________________ Phone ____________

Address ______________________________

City/State/Zip ______________________________

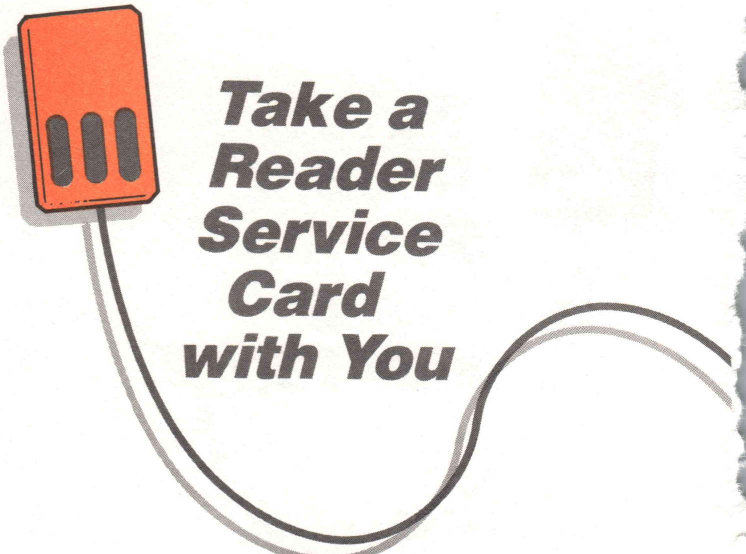
For Free Info ...

Start Here

Smart buyers start with *DDJ*'s free information card, a shopping center filled with information about the products and services advertised in this very issue: everything from software and systems to peripherals and professional support services.

And smart buyers can use this free information card to quickly and easily gather a comprehensive file of facts, figures and product specs to sort out competing claims. Using *DDJ*'s free information card can prevent you from making the wrong, costly buying decision.

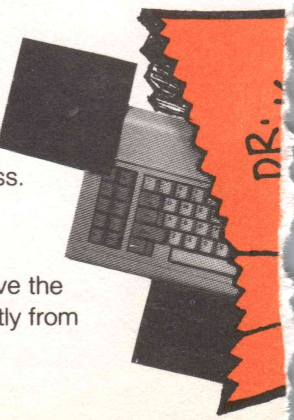
Be a smart shopper. Complete and mail this postage paid card today!



Take a Reader Service Card with You

It's Easy as ...

1. Circle the appropriate free information numbers, referring to the advertiser index for more information.



2. Fill in your name and address.

3. Mail today—postage is absolutely free. You'll receive the product information you need directly from the manufacturers, thanks to *DDJ*.

The Advertiser Index

*This advertiser prefers to be contacted directly; see ad for phone number.

Advertising Sales Offices

Southeast
Gary George (404) 378-1396

Midwest
Charlie Shively (415) 366-3600

Northern California/Northwest
Lisa Boudreau (415) 366-3600

Northeast
Cynthia Zuck (718) 499-9333
Martha Brandt (415) 366-3600

Southern California/AZ/NM/TX
Michael Wiener (415) 366-3600

Director of Marketing and Advertising
Ferris Ferdon (415) 366-3600

(continued from page 103)

the correct type and then fetch the object pointed to by *argp*, finally advancing *argp* past the object.

ANSI (and Unix V) have formalized the procedures I've just discussed into a set of macros. Examples 4 and 5, below, show *fang( )* rewritten in both ANSI and Unix forms. The ANSI form is, to my mind, more readable than is the Unix form. For one thing, I don't much like the Unix *va_dcl* macro, an invocation of which cannot be followed by a semicolon. The two sets of macros are more similar than not, however. In fact, *va_arg( )* is identical in both systems, and it is identical to the one I defined earlier. In the ANSI system, *va_list* is usually defined as *char* *, and the invocation:

```
va_start( argp, format )
```

usually expands to:

```
argp = (char *) &format +
                    sizeof(format);
```

That is, it initializes *argp* to point at the *format* argument and then advances *argp* past this argument to the next one on the stack. The Unix macros function in a similar manner.

```
#include <stdarg.h>              /*     ANSI     */

ansi_fang( format )
char    *format;
{
    va_list   argp;
    va_start( argp, format );

    for(;  *format ; format++ )
    {
        if( *format != '%' )
            putchar( *format );
        else
        {
            switch( *++format )
            {
            case 'c': printf("%c", va_arg(argp,int    ) ); break;
            case 'd': printf("%d", va_arg(argp,int    ) ); break;
            case 's': printf("%s", va_arg(argp,char *) ); break;
            case 'f': printf("%f", va_arg(argp,double) ); break;
            }
        }
    }
}
```

Example 4: *ANSI variable-argument conventions*

```
#include <varargs.h>             /*     UNIX     */

unix_fang( va_alist )
va_dcl                      /* NOTE: NO SEMICOLON PERMITTED HERE */
{
    char       *format;
    va_list   argp;
    va_start( argp );

    for( format = va_arg(argp,char*); *format ; format++ )
    {
        if( *format != '%' )
            putchar( *format );
        else
        {
            switch( *++format )
            {
            case 'c': printf("%c", va_arg(argp,int    )); break;
            case 'd': printf("%d", va_arg(argp,int    )); break;
            case 's': printf("%s", va_arg(argp,char *)); break;
            case 'f': printf("%f", va_arg(argp,double)); break;
            }
        }
    }
}
```

Example 5: *Unix variable-argument conventions*

Nifty Stuff

Curses

Last month's C Chest looked at a curses-subset package for the IBM PC. My implementation, however, lacked several useful features, such as overlapping windows and the ability to move or delete windows. It also couldn't handle various screen attributes and so forth.

I've just come across a very nice implementation of the full curses package done by Aspen Scientific. The package implements all of the Unix functions plus a few more that only work in the IBM environment—a hundred-odd functions in all. These extra functions give you more control over clearing the screen and let you work with screen attributes, change from the monochrome adapter to the CGA or EGA, scroll regions of a window, and so forth. The package also handles the IBM function keys in an intelligent manner. It can use direct memory mapping, the BIOS, and the ANSI.SYS driver for its output. A Unix-style manual is provided that's a considerable improvement on the real Unix documentation. There's a page for every function in the package, and each manual page contains a C example of how to use the function. Most important, the source code for the whole library is available. I don't use store-bought subroutine libraries unless I can get the sources because without them my programs cannot be ported outside the MS-DOS environment.

The product also comes with the source for a nifty screen generator called FAST that both provides an example of what the package can do and is a pretty useful program in its own right. FAST is a visual editor that lets you make data-entry screens interactively. It lets you define fixed text, the positions of various fields into which users can enter information, and the sequence of data entry. FAST generates a form-description file that is used by an interface-subroutine package, also provided, that lays on top of curses at run time.

MAKE YOUR 386 SWEAT LIKE A PIG.

Push every silicon sinew to its absolute limits with Real UNIX 386. It'll make your PC run like a racehorse. Unlike DOS for the antique 8088, and OS/2 which will someday be optimized for the 80286, UNIX 386 was built from the ground up specifically for the 386 by Intel, AT&T, Microport. It addresses every last bit in the 80386. All 32 of them. All today.

You get 80386 protected mode. And 80286 protected mode. And 8086 emulation. And network support. And 4 gigabytes of RAM including demand paged virtual memory. And the fastest 32-bit C, Fortran and Pascal compilers from Green Hills.** And dozens of UNIX applications. All right now so you can leap out ahead of those who wait. UNIX 386 running Microport DOSMERGE even gives you multi-user, multi-tasking PC/DOS. Up to 33 people can run applications like Lotus 1-2-3 and dBASE on low cost terminals without knowing or caring about UNIX.

All this from the people who bring you the world's fastest 286 UNIX for the IBM PC-AT. Need proof? Ask AIM Technology, the independent benchmarking service. They'll tell you that Microport UNIX screams past SCO XENIX 2.2, IBM XENIX System V, and the other UNIX System V look-alikes.†

Starting at only $199.

Honest to AT&T UNIX from Microport, the leader in logically priced UNIX and UNIX applications. If you want to squeeze every bit of performance out of the 386, and don't want to wait, call toll-free for complete information. But hurry, because until September 1st, every 40th caller mentioning this ad and our secret code will receive a complete UNIX 386 product. ABSOLUTELY FREE. ‡ What's the code? Just say "Unleash my 386." And give us a little squeal.

(800) 722-UNIX/(800) 822-UNIX in CA

Real UNIX:* $199

MICROPORT

Microport Systems, Inc.
10 Victor Square • Scotts Valley, CA 95066
(408) 438-8649 • Telex: 249554 MICR UR
FAX: (408) 438-2511

* A real Unix System, System V 386 $199 (2 user); Software Development System $399; Text Processing System $299; Complete System $799. ‡Limit one (1) call/winner per person/address. †AIM Technology benchmark, April 1987. **Green Hill Compilers 7209 Dhrystones (Interactive Systems PCC 4905 by comparison) UNIX is a trademark of AT&T. System V/AT and DOSMERGE are trademarks of Microport Systems, Inc. Other brands and products are trademarks of their respective holders.

CIRCLE 154 ON READER SERVICE CARD

C CHEST
(continued from page 106)

These routines let you get data from specific fields, validate data, and so forth.

Prices are $119 for an object-code-only version; $289 gets you the source code. A very stripped-down but adequate version of the Unix make utility is also provided. This is a nice product. If you need Unix-compatible screen output in your programs, or if you just want a nice clean window-management package, I'd recommend it. You can contact Aspen Scientific at P.O. Box 72, Wheat Ridge, CO 80034-0072; (303) 423-8088.

MiniProbe

It's no secret that one of the main strengths of Microsoft's C compiler is the Codeview debugger, and one of the main strengths of Codeview is the "tracepoint" (break when any of the following memory locations have changed) mechanism. For example, the command *TPB 0 52* forces a break when any of the bottom 52 memory locations in the data segment are modified, thereby finding where a "Null pointer assigment" actually happened in your program. A problem with tracepoints, however, is the amount of time it takes to process them. The region of memory is inspected in software after every instruction, and this inspection obviously takes a lot of time.

A solution to the problem is Atron's MiniProbe. The MiniProbe is a short-slot board that plugs into your computer. It provides you with four things: a hardware reset button, a "stop" button that generates an NMI (it can be used to break out of a loop when Codeview is ignoring Ctrl-Break), one hardware breakpoint, and one hardware tracepoint (that lets Codeview trace at full speed). At $395, the product is on the expensive side, but it really does work and is a godsend if you use tracepoints a lot. It's also a lot cheaper than a full hardware debugger. Contact Atron at 20665 Fourth St., Saratoga, CA 95070; (408) 741-5900.

It's no secret that one of Codeview's main strengths is the tracepoint mechanism.

Bug City

Gordon Arbuthnot found four bugs in the expression analyzer printed in the February C Chest (Listing Six, page 64):

1. Line 96: the declaration for *constant( )* is never used and can be deleted.
2. Line 112: A test for a blank space (*c==' '*) should be included here in case there's leading white space.
3. Line 231: The variable *tmp* should be declared as type *VTYPE* to avoid truncation of the intermediate results.
4. Lines 301–305: I forgot about engineering notation when I wrote this code, so the analyzer doesn't skip past stuff such as 100.5e+9 correctly. The corrected code is shown in Example 6, below. You should insert it in place of the code on lines 300 to 306 of the original listing (page 67). The line numbers in Example 6 reference the original listing.

I've also found two bugs in the priority-queue routines printed in the June 1987 issue. In *pq_ins( )* (Listing One, line 215) change:

```
memcpy( p->bottom += p->itemsize, &item, p->itemsize );
```

to:

```
memcpy( p->bottom += p->itemsize, item, p->itemsize );
```

and in *main( )* (Listing One, line 386) change:

```
i = pq_ins( queue, strsave(buf + 1) );
```

to:

```
p = strsave( buf + 1 );
i = pq_ins ( queue, &p );
```

```
298:    else
299:    {
            if( sizeof(VTYPE) != sizeof(double) )
                rval = (VTYPE) atol( Str );
            else
            {
                rval = atof( Str );

                while( isdigit(*Str) || *Str == '.' )
                    Str++;

                if(*Str == 'E' || *Str == 'e' )      /* 12.34E+03 */
                    Str += 2 ;
            }

            while( isdigit(*Str) )
                    Str++;
307:    }
```

Example 6: *Corrections to code in C Chest, Listing Six, February 1987*

Bibliography

Comer, Douglas. *Operating System Design, the Xinu Approach.* Englewood Cliffs, N.J.: Prentice-Hall, 1984. Pages 349–360 of this book present a version of *printf( ).*

Holub, Allen I. *The C Companion.* Englewood Cliffs, N.J.: Prentice-Hall, 1987. Stack frames, subroutine-linkage conventions, and the innards of *printf( )* are all discussed in depth in this book.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 5.**

Yes, You Should Get the Source Code

The Austin Code Works specializes in distributing C source code. There are four reasons why we believe you should get the source code of any program you license or purchase:

- *Understanding*

If you are going to make decisions or take action based on the output of a computer program then you need to be able to find out exactly what the program does with the data you give it. You should be able to study the algorithms used by a program to a level of detail which satisfies your perception of what you need to know. Your understanding shouldn't be constrained by what a documentation writer wants to tell you. There's only one way to build rock-solid confidence in the output of a computer program ... study and understand the source code.

- *Modification*

You shouldn't have to plead with the author or distributor of a program to fix bugs or make modifications that make the program more useful to you. There's only one way you can make a program do exactly what you want it to do and not do what you don't want it to do ... modify the source code.

- *Composition*

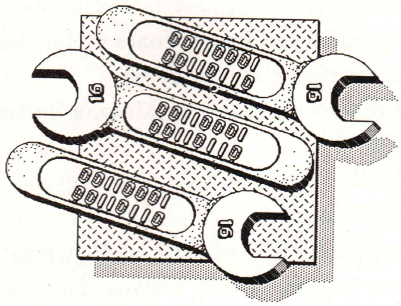
The bigger the blocks you build with the more quickly you can get done. The efficient and economical way to build large composite systems is to buy as many basic blocks as you can and write the glue code that makes them play together the way you want them to. There's only one form of code block that can quickly and easily be fitted to custom contours ... source code.

- *Evolution*

Any piece of code is just a starting point. Whether you run it stand-alone or include it in a larger project, you want it to grow with you and the system you are building. There's only one way to make a piece of code track your changing needs and requirements ... change the source code.

With these additional opportunities comes an additional responsibility. C is a very portable language but considerable freedom is given to C compiler writers. The result is that C compilers and C runtime libraries do differ. You should not expect a piece of C source code to compile and run right out of the box. It may, but you should be pleasantly surprised if it does and not dismayed if it doesn't. You may well have to lightly edit the code to satisfy the quirks and conventions of your favorite C compiler and its writer. Many seasoned programmers think this is a small price to pay for the enhanced value they get in return.

The Austin Code Works



by Ray Duncan

MS-DOS 3.30

Microsoft's OS/2 multitasking protected mode operating system and the IBM Personal System/2 computers captured most of the headlines on April 2 and afterward, but the upgrade to PC-DOS/MS-DOS 3.30 that was announced at the same time will have a more significant short-term impact on users. MS-DOS 3.30 is an evolutionary update from 3.20, upward compatible with previous versions, that incorporates bug fixes and some important enhancements. These enhancements fall into four general categories: new or improved user commands, configuration options, system functions, and device support. In addition, there is expanded "internationalization support" at many points throughout the system. The following information has been gleaned from the PC-DOS 3.30 users' manual, the technical reference, and a limited amount of experimentation.

New User Commands

A terminate-and-stay-resident MS-DOS extension, *NLSFUNC*, is a new element of MS-DOS internationalization support and allows new code pages to be selected for languages other than American English. *Code pages* are IBM's term for resident tables that define the mapping of character codes for the keyboard, list device, and video display.

Another TSR extension, called *FASTOPEN*, can be loaded that significantly improves performance for programs that repeatedly open and close a relatively small working set of files. *FASTOPEN* apparently caches the directory information for the most recently opened files and allows actual reads of the disk directory to be bypassed for files that are in the cache. *FASTOPEN* is active only for fixed disks and can support up to four drives; the default number of files cached per drive is 10, but a command-line option allows as many as 100 per drive to be cached.

The most interesting thing about *NLSFUNC* and *FASTOPEN*, when viewed in the light of the TSR-like *SHARE, GRAFTABL, GRAPHICS, KEYBXX, APPEND,* and *PRINT* commands, which were added in previous versions of MS-DOS, is the progressive trend toward decomposition of operating system functionality into independent, selectively loadable TSR modules. I hope we will see this trend continue and even accelerate in future versions of real mode MS-DOS—it helps minimize the squeeze on memory in 8086/88 machines and provides a welcome degree of flexibility.

Augmented User Commands

The *APPEND* command, which is a passive TSR that defines a search path for open operations on data files analogous to the *PATH=* command for executable and batch files, has been souped up slightly. New switches cause the *APPEND* path string to be stored in the environment block and allow the *APPEND* path to be searched on certain additional DOS function calls (*11h*, *4eh*, and *4bh*). *APPEND* was present in "generic" MS-DOS 3.20 but was previously distributed only with networking software in the IBM versions.

A BATCH file directive *(CALL)* has been added to allow one batch file to invoke another and then regain control without the intermediary of a secondary command processor, and the ability to use the name of an environment variable as a parameter inside a batch file (by framing it with % characters) has been documented at last.

A */S* switch has been added for *ATTRIB*, which allows the command to also be applied to matching files in all subdirectories of the named or default directory. *BACKUP* and *RESTORE* have new switches allowing selection of files by their date or time, and the new *BACKUP* can format disks on the fly (still can't begin to compare with *FASTBACK*, though). Other changes to *FORMAT* and *GRAPHICS* are too minor to discuss here. Finally, the *DATE* and *TIME* commands have been spiffed up so that they can reset the CMOS clock on PC/ATs (no more rooting around for your diagnostics disk with the SETUP program just to "spring forward" or "fall back").

New Configuration Features

The default value for *BUFFERS=* has been made a little "smarter." In previous versions, *BUFFERS=* always defaulted to 2. In MS-DOS 3.30, it defaults to a more appropriate value (in the range 2–15), depending on the type of disk the system is booted from and the amount of RAM installed.

(continued from page 112)

The *COUNTRY=* directive for CONFIG.SYS has been augmented with a code page option and with the ability to load internationalization information from a disk file. This capability works in concert with similar changes to the *KEYBXX* user command and will allow support for new date, time, and currency formats and collating sequences to be added much more easily. In the past, the internationalization support tables for various country codes were embedded inside the operating system; adding a new country code meant that the OEM had to rebuild the system files.

The *STACKS=* directive, which specifies the number of stack frames in the system pool for use by the interrupt handler, now behaves more sensibly. The default for the PC, PC/XT, and IBM Portable has been changed so that no stack switching occurs (the default for PC/ATs is still 9 stacks of 128 bytes each). Furthermore, users can always disable stack switching if desired by placing *STACKS=0* in the CONFIG.SYS file.

New System Functions

Two new system services are available for use by application programs, and the definition of *IOCTL* has been expanded slightly. *Int 21h,* function *67h* (Set Handle Count) allows the file table for a process to be expanded so that the process can have more than 20 files open at once. This subject was nearly beaten to death in this column about a year ago. Most of you will no doubt remember that in MS-DOS, Versions 3.0–3.2, there is a 20-byte table corresponding to file handle numbers in a reserved area of the PSP along with a double-word pointer to the table and an additional word that gives the length of the table. Plenty of *DDJ* readers have already hit on the fact that you can get around the 20-file limit simply by building a new, larger table somewhere and modifying the PSP words containing the pointer and length accordingly. Apparently, the new function call works in about the same way. It seems to allocate a block of memory outside the process itself that is large enough to hold the expanded table, copies the old file table to the new one and initializes the as-yet-unused positions, and then twiddles the PSP to point to the new table.

Int 21h, function *68h* (Commit File) forces all the internal disk buffers associated with a file handle to be written to disk and the directory information for the file to be updated. This is effectively the same as DUPing the handle for a file and then closing the new handle, except that the DUP method can fail if the system is out of handles.

The calling sequences for the new functions *67h* and *68h* are summarized in Table 1, below.

Int 21h, function *44h (IOCTL)* has a new subfunction (*0ch*) that allows an application program to select a different code page for a peripheral device. This is simply another facet of the expanded internationalization support.

Device Support

MS-DOS 3.30 supports double-sided, double-density, 1.44-megabyte, 3.5-inch disk drives and the new higher-resolution video adapters found on the IBM Personal Computer/2 models. The built-in asynchronous communications driver has been expanded to support up to four serial ports and presumably is now interrupt-driven because the documentation

```
Set Handle Count:
Int 21h, function 67h

Call with:  ah = 67h
            bx = desired number of handles

Returns:    Carry clear if function succeeded
            or
            Carry set if function failed and
            ax = error code


Commit File:
Int 21h, function 68h

Call with:  ah = 68h
            bx = file handle

Returns:    Carry clear if function succeeded
            or
            Carry set if function failed and
            ax = error code
```

Table 1: *New function calls in PC-DOS/MS-DOS 3.30*

(continued from page 114)

claims it can handle data rates as high as 19,200 baud.

Prices

PC-DOS 3.30 costs $120 new or $75 as an update (you have to send the cover page from the users' manual of a previous version of PC-DOS along with the payment to qualify for the update). Both a 5¼-inch and a 3½-inch disk are included in the package. The *DOS 3.30 Technical Reference* costs $85. The executable files for the linker, DEBUG, and EXE2BIN along with the source file for the VDISK driver are no longer supplied on the PC-DOS distribution disks but are on a disk that comes with the technical reference instead.

32-Bit Book Nook

Back when I started dabbling in 8086 programming, the Intel manuals were poorly written and indexed and even more difficult to use than they are now. I eventually discovered Rector and Alexy's *The 8086 Book*, to my immense relief, and have well-worn copies of it stashed everywhere I work. Although *The 8086 Book* has a few weaknesses, they are far outweighed by the accuracy of the book and the organization and presentation of the information about the 8086 instruction set.

Rector and Alexy's book is looking pretty dated these days, though, what with the many new instructions, exceptions, and protected mode addressing considerations of the 80286 and 80386. I keep hoping that someone will publish an equally useful book that experienced programmers can use as a reference to the entire family of Intel 80x86 processors, but thus far no new trade book has filled the bill. The Intel manuals have improved by leaps and bounds, and in their latest incarnations are real treasure troves (though still somewhat dense: every word is significant). But the authors of most assembly-language books seem content to provide rehashed and weakened versions of the Intel manuals—they seem to have lost the concept of added value altogether.

Programming the Intel 80386, by Smith and Johnson[1], at first glance looked like a possible successor to Rector and Alexy's book. The obligatory 20 pages of explanation of bits and bytes are followed by a brief look at the 80x86 processor line; a review of 80386 registers and addressing modes; an overview of the 80386 instruction set by functional group; and then the body of the book: a 180-page, alphabetically organized reference to the 80386's instructions, including opcodes, clocks, flags affected, notes, and examples, with each instruction beginning on a new page. The last 65 pages of the book contain a rather sketchy overview of protected mode segmentation, virtual memory, paging, caching, and even a few words about 80386 bus signals.

I really wanted to like this book, but it is just too uneven. The organization (other than the main reference section) needs improvement, there are literally no programming examples, and some crucial subjects (such as the distinctions between segmented virtual memory, paged virtual memory, and segmented paged virtual memory) just aren't covered in adequate depth. But the main problem with the book is that the authors are clearly paraphrasing their material from other sources instead of writing from a solid base of 80386 programming experience. Without this experience, they simply do not have a sense of which material is useful and how it should be presented and which material (such as the 80386 bus signals) is fluff and should be omitted.

We are now 0 for 2 on 80386 assembly-language books reviewed in this column (*80386/80286 Assembly Language Programming* by Murray and Pappas, reviewed in the March 1987 issue of *DDJ*, didn't cut the mustard either). At present, if you are interested in the 80386, your best value is still the *Intel 80386 Programmer's Reference*, which is far more authoritative, readable, and thorough than any of its trade book competitors--and cheaper besides.

Word Meets Its Match

Fred Heutte, of Portland, Oregon, writes: "I'm turning in the extra-credit question for my final exam in

Algorithm Design 101 (responding to Larry Heberlein in the March *DDJ*).

"I am familiar with the search-and-replace problem in Microsoft Word. The particular example Larry Heberlein uses is a good one—it's virtually a worst case for Word, which is paragraph-oriented. When you remove the paragraph marks, all kinds of contortions result to keep track of the conversion. Other than this particular case, Word's search-and-replace is very nice, and the Bellevue gang deserves applause for this as well as many other Word features.

"I find it hard to believe that anyone can beat my entry, however. I am running INFORMIX-SQL, Version 2.0.0 on an AT&T 6300-Plus. Informix has a program called SFORMBLD that compiles screen entry forms to be run under the data entry form module SPERFORM. When compiling a form file of no more than 11K, an 'out of memory' error is returned. With DOS 3.1 taking about 40K RAM and SFORMBLD about 110K, this leaves about 490K—an approximate ratio of 44.5:1.

"No wonder I'm pushing the organization I work with to convert to Rbase System V, even though it's not SQL!"

Another Reply to Mr. Lyall

Mr. Davidson Corry, of Seattle, Washington, writes: "I read with some interest Mr. Charles Lyall's letter to your column in the March issue on the continuing thrash between proponents of assembly and high-level languages.

"As it seems to be customary to establish one's pedigree, let me offer you mine. I got my start on a Singer process-control machine (vintage early 50s) with 4K of magnetic drum memory in 1965. 'Assembly' language at its best—bootstrapping hex codes off paper tape! Since then I've used a dozen-odd other languages on two-dozen other machines, finally curling comfortably up beside an AT clone with MASM and C, consulting on systems programming for companies in the Seattle area.

"Mr. Lyall computes that a 10:1 execution speed ratio is compensated forby the 1:8 productivity increase of writing in a high-level language; that he would have to 'run the little turkey 700 times' to break even. I picture him sitting calmly at his terminal, waiting for a compile to complete, serene in the knowledge that the author of the compiler has worked productively. This is a level of rationality I have not reached and to which I do not aspire. Not me, babe: I want that sucker linked and up! Now!

"I have equally cursed the nameless sculptors of silicon who—for perhaps excellent engineering reasons—hobbled the 8086 with short-range conditional jumps and saddled programmers with the jump around a jump. However much it simplified their job, it has made mine that much more difficult.

"This is, I think, the central issue in the language-level debate: the ultimate judgment on software is on its effectiveness (speed, power, features, and so on) as a tool for end-users. The difficulties of engineering the tool are, in the end, irrelevant. The only justification, and it is a weak one, for taking 'shortcuts' is that a good tool today is more useful than a superb tool next year.

"To my mind, there are only two values of execution time: 'fast enough not to notice' and 'slow enough to get impatient.' No one has reasonably suggested that a well-coded, high-level program can beat a well-coded, assembly-language one for time, and my experience suggests that assembly-language code is much more likely to fall on the good side of the 'come on!' threshold. So I often write in assembly language, for speed, compactness, and close control of the hardware. It's not portable, you say? Yes, but a program that works with my PC keyboard and screen probably isn't going to fit in the CICS mindset anyway.

"Having said that, I will tell you that I much prefer high-level languages and use them whenever I can get away with it—that is, whenever the hardware penalty is not too dear. This does not contradict my argument—it confirms it. Let me explain.

"Assembly language, however efficiently it tickles the chip, is a pain to write and debug. It is cryptic, tedious, verbose—even COBOL is better! By its very flexibility it encourages all sorts of 'gimmick' coding ('Fly

(continued from page 117)

now, crash later . . . '). And the typical assembly-language source listing almost totally obscures the underlying algorithm.

"In contrast, my favorite language these days is ICON. It is no speed demon, and makes no claims to be, but it is a superb notation for recording an algorithm. The most difficult part of learning ICON seems to be unlearning coding practices that got you around limitations in other languages so that you can see the problem fresh and use the power of ICON expressions.

"C—the 'portable assembly language'—was designed to let programmers access a pervasive architecture efficiently: linear/rectilinear arrays of machine objects (bytes/words/longs/floats/doubles/pointers) or agglomerations of them. The 'overhead' of compiled C varies as the CPU architecture drifts from this ideal, but C may be near ultimate in letting programmers talk the chip's language comfortably.

"Does anyone remember UCSD Pascal, running on a p-machine? No, not the software emulator—I mean the real Western Digital p-machine in silicon. It screamed, or so they said. How about Modula-2 on Wirth's Lilith? And whatever happened to the Forth chip set that executed Forth primitives directly? These are attempts to make the chip talk the programmer's language.

"And that brings us full circle. Energy spent engineering hardware solutions is repaid a thousandfold at the next higher level—because that many more people use it. If a thousand people run Mr. Lyall's 'little turkey,' every one of them loses ground on the very first run.

"The 'debate' over high-level-language efficiency is an artifact of adapting human language to machine architecture, a historical accident. I suggest that a better solution is to design notations in which humans can clearly and conveniently express algorithms and then adapt the 'hardware' (RISC chips with a microprogrammed icing, better compilers and operating systems, and so on) to execute these improved notations efficiently."

Mahalo and Aloha

Every good thing must come to an end, and although I have enjoyed writing this column and have learned far more from *DDJ's* readers than they have learned from me, five years of a Good Thing is definitely enough. The siren song of the fabulous new class of 32-bit personal computers, like the Mac II and the PS/2 Model 80, is becoming irresistible, and I've got a lot of delving to do before I can write about those machines intelligently. In the meantime, I wish you all continued health, prosperity, happiness—and a machine with a BIG fixed disk, tape backup, and no-wait-state RAM!

Note

1. Bud E. Smith and Mark T. Johnson, *Programming the Intel 80386* (Glenview, Ill.: Scott, Foresman & Company, 1987. 346 pages including index. $22.95. ISBN 0-673-18568-0.

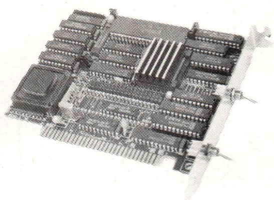
DDJ

Vote for your favorite feature/article.
Circle Reader Service **6**.

MICROWAY ACCELERATES YOUR PC!

FastCACHE-286™

Runs your PC Faster than an AT!
Runs the 80286 at 9 or 12 MHz and the 80287 at 8, 9 or 12 MHz. Includes 8 kbytes of 55ns CACHE.



Compatible with IBM PC, XT, Leading Edge Model D, Compaq, and Turbo motherboards. Includes 8088 Reboot Switch, DCache, Print Spooler and Diagnostics .... **From $399**

LOTUS/INTEL EMS SPECIFICATION BOARDS

MegaPage™ The only EMS board which comes populated with two megabytes of cool-running, low power drain CMOS RAM installed. Includes RAM disk, print spooler, disk cache and EMS drivers. For the IBM PC, XT and compatibles... **$549**

MegaPage with ØK **$149**

MegaPage with 2 megabytes of HMOS RAM **$419**

MegaPage AT/ECC™ EMS card for the PC AT and compatibles includes Error Correction Circuitry. With ECC, 11 RAM chips cover 256K so the user never encounters RAM errors. With 1 megabyte CMOS RAM **$699**

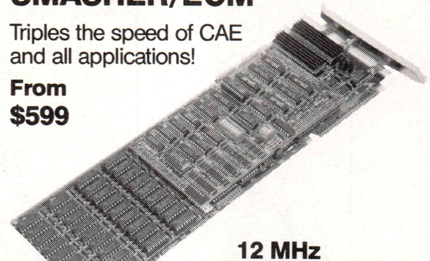
INTEL, JRAM, or Maynard .... CALL

INTEL INBOARD 386 ØK $1250

NUMBER SMASHER/ECM™

Triples the speed of CAE and all applications!

From $599

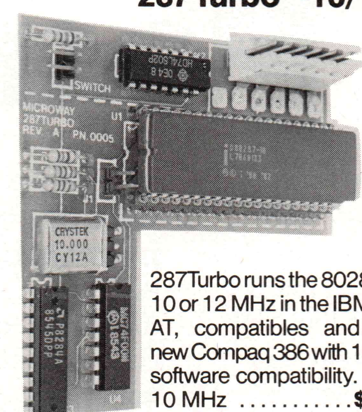


12 MHz 8086/8087 Accelerator Plus
A Megabyte for DOS!
For the IBM PC, XT and compatibles
***PC Magazine* "Editor's Choice"**

8087 SOFTWARE

IBM BASIC COMPILER	**$465**
MICROSOFT QUICK BASIC	**$79**
87BASIC COMPILER PATCH	**$150**
87BASIC/INLINE	**$200**
IBM MACRO ASSEMBLER	**$155**
MS MACRO ASSEMBLER	**$99**
87MACRO/DEBUG	**$199**
MICROSOFT FORTRAN V4	**$299**
RM FORTRAN	**$399**
LAHEY FORTRAN F77L	**$477**
MS or LATTICE C	**CALL**
STSC APL★PLUS/PC	**$450**
STSC STATGRAPHICS	**$675**
SPSS/PC+	**$695**
87SFL Scientific Functions	**$250**
87FFT	**$200**
OBJ → ASM	**$200**
PHOENIX PRODUCTS	**CALL**

287Turbo™-10/12



287Turbo runs the 80287 at 10 or 12 MHz in the IBM PC AT, compatibles and the new Compaq 386 with 100% software compatibility.

10 MHz **$450**
12 MHz **$550**

***PC Magazine* "Editor's Choice"**

8087 UPGRADES

All MicroWay 8087s include a one year warranty, complete MicroWay Test Program and installation instructions.

8087 5 MHz.................... $99
For the IBM PC, XT and compatibles

8087-2 8 MHz $154
For Wang, AT&T, DeskPro, NEC, Leading Edge

80287-3 5 MHz............... $159

80287-6 6 MHz............... $179
For 8 MHz AT and compatibles

80287-8 8 MHz............... $259
For the 8 MHz 80286 accelerator cards

80287-10 10 MHz............ $395

80387-16 16 MHz............ $495

PC-PAL™ Programmer...... $395

64K 150ns ..... $15   256K 150ns $36
Call for great prices on V20 & V30

AT8™

Turns your AT into a high speed, multi-user Xenix business system!

8 port, intelligent serial controller with 3% response degradation. Includes 8 MHz 80186 with built in DMA **$1299**

MICROWAY SOFTWARE FOR LOTUS 1-2-3™

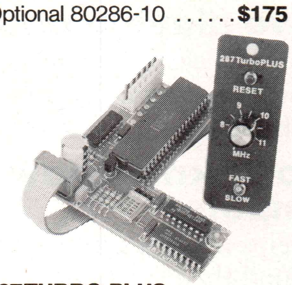
PowerDialer® Add-In for Lotus 1-2-3 Release 2. Automated telephone dialing from within 1-2-3. Adds least cost routing, automatic carrier selection and automated phone book worksheet. Builds customized dialing applications. Can be used with DesqView **$79**

FASTBREAK™ employs the 8087 to increase the speed of Lotus 1-2-3™ Version 1A or 1A*. Users are reporting speed ups of between 3 and 36 to 1. When run with our NUMBER SMASHER accelerator card, recalculation speed ups of 10 to 30 are being reported **$79**

HOTLINK™ adds easy linking of spreadsheets to Lotus 1-2-3 Version 1A... **$99**

287 TURBO-PLUS™

Speeds up your AT
Adjustable 80286 Clock 6-12 MHz
10 MHz 80287 Clock
Plus Full Hardware Reset **$149**
Optional 80286-10 ...... **$175**



287TURBO-PLUS
With 80287 10 MHz **$549**
With 80287 12 MHz **$629**

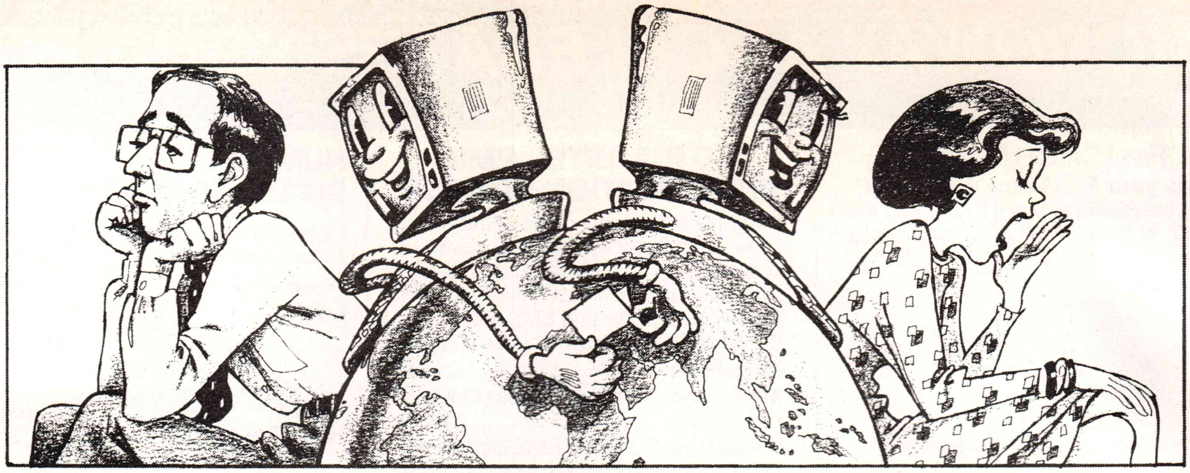
CALL (617) 746-7341 FOR OUR COMPLETE CATALOG

MicroWay
P.O. Box 79
Kingston, Mass.
02364 USA
(617) 746-7341

The World Leader in 8087 Support!

MicroWay Europe
32 High Street
Kingston-Upon-Thames
Surrey England KT1 1HL
Telephone: 01-541-5466

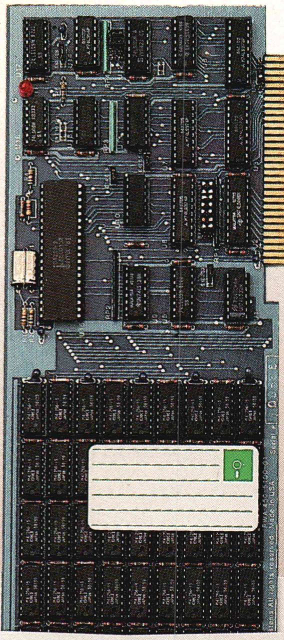
CIRCLE 300 ON READER SERVICE CARD



Think fast! Pick the better fit...

Our 5th Year Bonus!
Mention this ad when ordering
and get your choice of a V 20-8
(replaces 8088) or $20 off on
your Battery Backup purchase!

FLOPPY DISK.
- Fills time between coffee breaks
- Makes a hard disk seem *fast*
- Your computer appears busy (even if you aren't!)
- Wears out moving parts



SEMIDISK Disk Emulator.
- Gets that job done NOW
- Makes a hard disk seem *slow*
- Maximizes your productivity with anything from databases to compilers
- Totally silent operation

...for YOUR demanding tasks.

SURPRISE! *Neither* is memory mapped, so they don't affect your precious Main Memory. *Both* retain data indefinitely - even with the computer turned off.

THE SEMIDISK SOLUTION. You could invest in a series of "upgrades" that turn out to be expensive band-aids without solving your real problem. Even those "Accelerator" and "Turbo" boards do little to speed up disk-bound computers. If your applications spend too much time reading and writing to disk (and whose don't?), you won't want to settle for anything less than a SemiDisk disk emulator. The SemiDisk comes in 512K and 2Mb capacity. More boards may be added to make up to an 8 Megabyte SemiDrive!

SPEED THAT'S COMPATIBLE. PC, XT or AT, if you need speed, the SemiDisk has it. How fast? Recent benchmarks show the SemiDisk is from 2 to 5 times faster than hard disks, and from 25% faster (writing) to several times faster (random reads) than VDISK and other RAMdisk software that gobble up your main memory.

MEMORY THAT'S STORAGE. Using our small external power supply, with battery backup, your data remains intact through your longest vacation or even a seven-hour power failure!

CELEBRATE WITH US! Now, SemiDisk celebrates its fifth birthday with a special offer for IBM-PC owners. Buy a SemiDisk now and we'll include an 8 MHz V-20 microprocessor (replaces the 8088) to make your new SemiDisk run even faster. Don't need the V-20? We'll take $20 off the price of your Battery Backup Unit!

	512K	2Mbyte
IBM, PC, XT, AT	**$495**	**$ 795**
Epson QX-10	**$495**	**$ 995**
S-100 SemiDisk II	**$795**	**$1295**
S-100 SemiDisk I	**$299**	-------------
TRS-80 II, 12, 16	**$495**	**$ 995**
Battery Backup	**$130**	**$ 130**

Someday you'll get a SemiDisk.
Until then, you'll just have to...wait.

SemiDisk

MasterCard VISA

SemiDisk Systems, Inc., 11080 S.W. Allen Blvd., Beaverton, Oregon 97005 (503) 626-3104

CIRCLE 85 ON READER SERVICE CARD

Translating from MS-BASIC to C

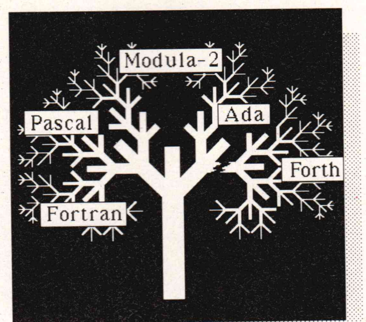
by Namir Clement Shammas

In this issue I discuss translating programs from BASIC to C. This article is the third in a series that looks at translating programs between different languages or dialects of the same language. As in the previous articles in the series, I have used a language translator. I will also discuss the translation performed by BASTOC (Version 2.1), a package from JMI Software (P.O. Box 481, Spring House, PA 19477). The version I used converts MS-BASIC source code to C source code compatible with Microsoft C, Version 4.0. Versions of BASTOC are available to translate from several other BASIC dialects, such as CBASIC, too.

To guide the discussion, I have selected three programs: the Sieve benchmark, a root-seeking program, and a multifile find/replace utility. I have actually translated many other BASIC programs to test how well BASTOC works, and the results have shown that the package's source code conversion is well thought out and comes close to being a complete translation. The converted programs kept much of their feel but ran much faster.

The Sieve Benchmark

Listing One, page 86, shows the BASIC source code for the Sieve benchmark. The program demonstrates the conversion of arrays, *FOR* and *WHILE* loops, and *IF* statements. Listing Two, page 86, shows the C version of the BASIC source code. The first BASIC statements are *OPTION BASE* and *DEFxxx*. BASTOC ignores *OPTION BASE 1* declarations because the C language implicitly uses 0 as the lower array bound. Thus, *OPTION BASE 1* creates a bit of wasteful space when translated. BASTOC uses *DEFxxx* statements well to assign a data type to the BASIC variables. Because the BASIC Sieve program makes the *DEFINT A_Z* declaration, the C version declares all the inherited variables as *int*. Notice that the array flag is replaced with a pointer to integers, **FLAGS*, because the dimensioning of the array flags uses a variable and not an integer constant. This dictates the use of dynamic allocation and pointers. The BASTOC translator declares additional identifiers that it uses in controlling loops. The simple BASIC assignments are converted in a straightforward fashion. *DIM FLAGS (SIZE)* yields a dynamic allocation of the array using the *balloc* function.



The few BASIC *PRINT* statements are not converted into *printf* function calls, as you might have expected. Instead, BASTOC uses its own overloaded *BPRINT( )* function (which I presume is more efficient than *printf*). Examine the three *BPRINT* calls in the C code and notice the first argument. It is a string that maps the number and type of data to be displayed. The letters *s* and *i* indicate a string and an integer, respectively. The function *BPRINT* is able to tackle a varying number of arguments.

The manipulation of time is carried out using a function that sets the time—namely, *STIME_( )*—and another—*TIME_( )*—to return it.

Because C has more options than MS-BASIC does, translating the *FOR* and *WHILE* loops is easy. Notice that the C version uses additional identifiers to define the iteration range of the *FOR* loops. The single *IF* statement that uses a *GOTO* in its *THEN* clause is converted into a similar set of statements. The BASTOC translator makes use of the available *gotos* in C rather than trying to alter the program flow by using other constructs. You can replace the use of the *gotos* with more structured *if . . . else* clauses if you are willing to hand code the changes with your editor.

The Root-Finding Program

The second program, which finds the roots of a single nonlinear function, is in Listing Three, page 87. It has the following features:

- Using the *ON GOSUB* statement, the program can look at multiple functions.
- The accuracy and maximum number of iterations are preassigned. If the number of iterations exceeds the maximum limit, the accuracy is relaxed.

This program demonstrates the conversion of BASIC *DATA*, *ON GOSUB*, *GOSUB*, and *PRINT USING* statements.

Listing Four, page 87, shows the translated C code. The first declaration tackles a data structure that is employed in converting BASIC *DATA* statements. The C structure is composed of an unsigned integer and a pointer. The integer stores the original BASIC line number, and the point-

USE THE BRAINS YOUR IBM WASN'T BORN WITH.

Right at your fingertips in CompuServe's IBM® Forums.

Our IBM Forums involve thousands of users worldwide who will show you just how easy it is to get the most from your IBM and IBM compatibles.

The IBM New Users Forum lets you ask basic questions of PC experts. The IBM Junior Forum is perfect for PCjr® users. Trade tips with other IBM PC and AT users in the IBM Software Forum. Ask questions and get answers directly from the manufacturers in the PC Vendor Support Forum. And if you're looking for a PC Bulletin Board, visit the IBM Communications Forum. Or try the IBM Hardware Forum for discussions on hardware topics and product updates.

Easy access to free software, including free uploads.

You can easily download first-rate, non-commercial software and utility programs. Upload your own programs free of connect time charges. And take advantage of CompuServe's inexpensive weeknight and weekend rates, when forums are most active and standard online charges are just 10¢ a minute. You can go online in most areas with a local phone call. Plus, you'll receive a $25.00 Introductory Usage Credit when you purchase your CompuServe Subscription Kit.

Information you just can't find anywhere else.

Use the Forum Message Board to send and receive electronic messages. Join ongoing, real-time discussions in a Forum Conference. Communicate with industry experts, including the programmers who write your favorite programs. Search Forum Data Libraries for non-commercial software and shareware. Enjoy other useful services too, like electronic editions of popular computer magazines.

All you need is your IBM computer or IBM compatible computer (or almost any other personal computer) and a modem.

To buy your Subscription Kit, see your nearest computer dealer. Suggested retail price is $39.95. To receive our free brochure, or to order direct, call 800-848-8199 (in Ohio and Canada, call 614-457-0802). If you're already a CompuServe subscriber, type GO IBMNET (the IBM Users Network) at any ! prompt to see what you've been missing.

CompuServe®

Information Services, P.O. Box 20212
5000 Arlington Centre Blvd., Columbus, Ohio 43220
800-848-8199
In Ohio, call 614-457-0802
An H&R Block Company

CIRCLE 237 ON READER SERVICE CARD

(continued from page 122)

er refers to a string containing the information contained in a BASIC *DATA* statement. The original BASIC line numbers are maintained in the data structure to accommodate the *RESTORE <line number>* statement in BASIC. The transformed program also declares the absolute, exponential, and power functions that must be called from C libraries.

Notice that the first two lines of the BASIC program are the *DEFDBL* and *DEFINT* declarations. Line 1010 was originally *DEFDBL A_Z*, but the translator objected to the redefinition of the I/O range in line 1020. All the BASIC variables except the integer flag *Diverge%* adhere to the default type-by-name association. The BASTOC translator handles the data typing of variables correctly. Notice that the BASIC *Diverge%* variable is converted into *divergeI* in C: the uppercase *I* replaces the % character in BASIC.

The BASIC *INPUT* statement is converted into a call to an overloaded C function, *INPUT( )*. Like the *BPRINT( )* function discussed earlier, the first argument of *INPUT( )* is a string that seems to determine if a prompt is used as well as to indicate the data type of the input. Like the familiar *scanf( )* function, *INPUT( )* uses the address of the variable to store the input data. The BASIC *READ* statements are translated into function calls that are similar to the *BPRINT( )* and *INPUT( )* functions.

Translating the BASIC lines that make up the iteration loops takes place without snags. Looping with the *WHILE-WEND* and the *IF* statements is correctly converted. The *GOSUB 1200* statements are replaced with calls to function *pr_1200( )*. Because *GOSUB*s take no parameters, their counterparts in C are always parameterless functions. If you use BASTOC to translate your own BASIC programs into C, you may want to edit your program to take advantage of using argument lists in C functions.

PRINT USING statements are translated into *UPRINT( )* function calls. These C functions resemble their *BPRINT( )* cousins, except the first argument is the output format string and the second argument contains the number and data types for the output variables.

The MS-BASIC subroutine that starts at line 1200 uses the *ON GOSUB* statement to call other subroutines. The *ON GOSUB* is translated into the C switch-case decision-making construct. The variable used in selecting the proper case is the global *n* identifier. The C function *pr_1200( )* ends with a *return* with no expression associated with it. The same is true for the rest of the subroutines.

The Find/Replace Utility

The third example program is shown in Listing Five, page 89, and Listing Six, page 90, shows the C version. The BASIC program performs find/replace operations on one or more files. This program demonstrates the translation of sequential file I/O, string manipulation, and error handling.

In comparing the BASIC declarations in lines 1040 to 1070 with those in the C version, notice the following:

- The effect of *OPTION BASE 1* is ignored, and BASIC dimensions of 20 and 30 elements are replaced with 21 and 31 elements. The first array elements, with an index of 0, are not used.
- *DEFINT A_Z* is used to tell BASTOC how to declare the data types of scalar variables in the C program. The underscore character replaces the dot used in the BASIC variable names.
- String-type scalars and arrays are declared as pointers.

The first set of statements inside the *main( )* function allocates space to the string arrays using the corresponding pointers. The assignments to integer variables are straightforward. String assignment involves a call to function *s_asgn( )* instead of an ordinary assignment as in BASIC, Turbo Pascal, or Modula-2.

What is even more unusual is the way the *GOSUB 2290* is handled: instead of yielding a call to function *pr_2290( )*, two statements are used. The first is *sub_push(1)* and the second is a *goto*. Has the BASTOC translator finally collapsed under pressure and continuous use? No, there is no need to panic! This sort of code seems to be generated when the BASIC *ON ERROR* is used. If you look at the C label *L_2290*, you see a *goto sub_ret*, which directs the program flow to the correct label. Although this may remind you a bit of spaghetti BASIC code, no one said that translating error handling from one language to another was easy! The large number of labels and *goto*s is the result of supporting BASIC's error handling. The BASTOC translator resorts to defensive programming to cover any errors generated from numerous lines. Three additional code segments are inserted by BASTOC to handle errors, each starting with a label. The first is *sub_ret*, mentioned earlier, which is responsible for simulating *gosub returns* when error handling is used. Label *err_trap* is where the program flow first resumes after an error occurs. From label *err_trap* the program flows to the error-handling routine inherited from the BASIC source code. In this example, the BASIC error handler starts at line 1750, and consequently, the C version resumes at label *L_1750*. Notice that the transformed BASIC error-handling code ends with a *goto un_trap* to resume program execution. The code segment following the *un_trap* label contains a switch-case with a long list of *case* clauses to direct the program resumption.

File I/O operations in the C version resort to calling several functions that emulate their corresponding BASIC statements. These include functions *BOPEN( )*, *BCLOSE( )*, *INPUT( )*, and *BPRINT( )*. The use of the last two functions has been extended to include file I/O. The first two arguments of *INPUT( )* are a string-type file I/O indicator and the buffer number. The *BPRINT( )* function call performing file output differs from the one involved in displaying a variable by having the buffer number as the first argument. The rest of the arguments are the same, as you might expect. The BASIC *LPRINT* statements are replaced with calls to the C function *BLPRINT( )*. The *BLPRINT( )* function is to the *BPRINT( )* function as BASIC's *PRINT* is to *LPRINT*.

String manipulation involves calls to functions that clone BASIC string functions, such as *MID$( )*, *LEN( )*, and *INSTR( )*. String concatenation employs the function *s_asgn( )*, following typical methods used in C for string management.

STRUCTURED PROGRAMMING
(continued from page 124)

Summary

Converting interpreted MS-BASIC programs to run as compiled C programs has limitations. Many of these limitations exist because of the compiled nature of the functioning C versions—for example:

- *CHAIN* is translated by BASTOC into a program execution that doesn't return to the original program.
- *COMMON* declarations must be identical in the chained BASIC programs.
- *PEEK*s, *POKE*s, *VARPTR*, and *USR* are not supported for the C and L memory models.

Translating BASIC programs into C enables your programs to gain the speed of compiled programs. The alternative is to use a BASIC compiler to attain the desired speed. Why go the C route instead of the QuickBASIC or Turbo BASIC way? Transporting BASIC programs to C for the sole purpose of gaining speed may be overkill. The advantage of translating BASIC programs is that C programs can be modified to use more popular libraries, and more important, they can be enhanced by linking them with third-party C libraries. This enables you, for example, to incorporate more elegant windowing routines in your programs. Also, your programs will be able to make use of C's efficient pointer manipulations, data structures, enumerated types, unsigned integers, long integers, and more.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listings begin on page 86.)

Vote for your favorite feature/article.
Circle Reader Service **No. 7**.

EVEN MORE POWER AND FLEXIBILITY

BRIEF 2.0

Users and industry press alike have unanimously proclaimed BRIEF as the best program editor available today. Now, the best gets better, with the release of BRIEF 2.0.

Straight from the box, BRIEF offers an exceptional range of features. Many users find that BRIEF is the only editor they'll ever need, with features like real, multi-level Undo, flexible windowing and unlimited file size. But BRIEF has tremendous hidden power in its exclusive macro language. With it, you can turn BRIEF into your own custom editor containing the commands and features you desire. It's fast and easy.

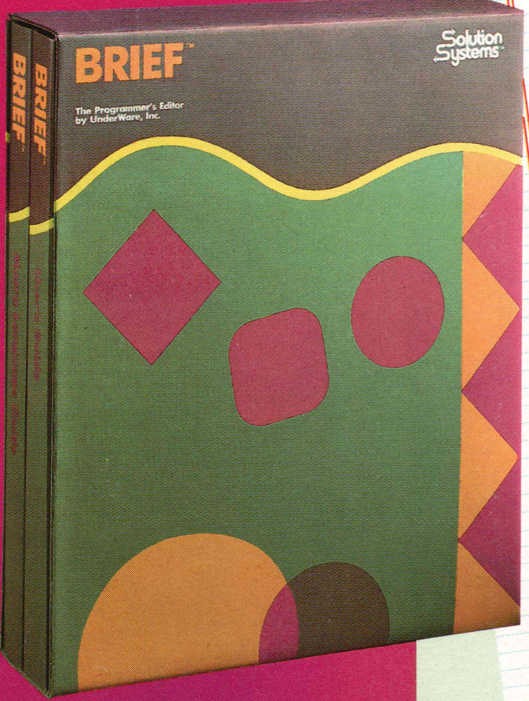
Jerry Pournelle, columnist for BYTE magazine summed it all up by saying BRIEF is, "**Recommended. If you need a general purpose PC programming editor, look no further.**" His point of view has been affirmed by rave reviews in C JOURNAL, COMPUTER LANGUAGE, DR. DOBB'S JOURNAL, DATA BASED ADVISOR, INFOWORLD AND PC MAGAZINE.

One user stated "BRIEF is one of the few pieces of software that I would dare call a masterpiece." Order BRIEF now and find out why. **BRIEF 2.0 is just $195. If you already own BRIEF, call for upgrade information.**

TO ORDER CALL: 1-800-821-2492
(in MA call 617-337-6963)

As always, BRIEF comes with a 30 day money-back satisfaction guarantee.

Solution Systems™
541 Main Street
Suite 410D
So. Weymouth, MA 02190
(617) 337-6963



Look at these BRIEF 2.0 enhancements!

Main Features:

- All new documentation with tutorials on basic editing, regular expressions **and** the BRIEF Macro Language.
- Setup program for easy installation and configuration. (Requires no knowledge of the macro language)
- Increased speed for sophisticated operations like Undo and Regular Expression Search.
- Expanded regular expressions, with matching over line boundaries.
- More block types, with marking by character, line or column.
- Command line editing (move cursor, add and delete characters, specify command parameters).
- Support for more programming languages.
- Optional borderless windows.
- Enhanced large display support, including wider displays.
- Reconfigurable indenting for C files (supports most indenting styles).

Plus the basic features that made BRIEF **SO** popular!

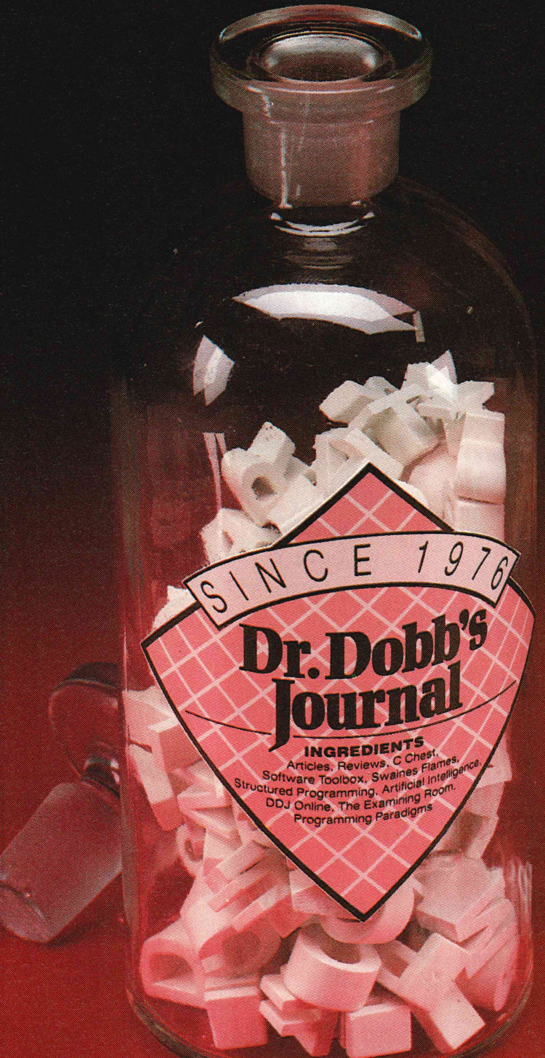
Basic Features:

- Full multi-level Undo
- Windows
- Edit many files at once
- File size limited only by disk space
- Automatic language sensitive indentation

Requires an IBM PC or compatible with at least 192K RAM.
BRIEF is a trademark of UnderWare, Inc.
Solution Systems is a trademark of Solution Systems.

CIRCLE 142 ON READER SERVICE CARD

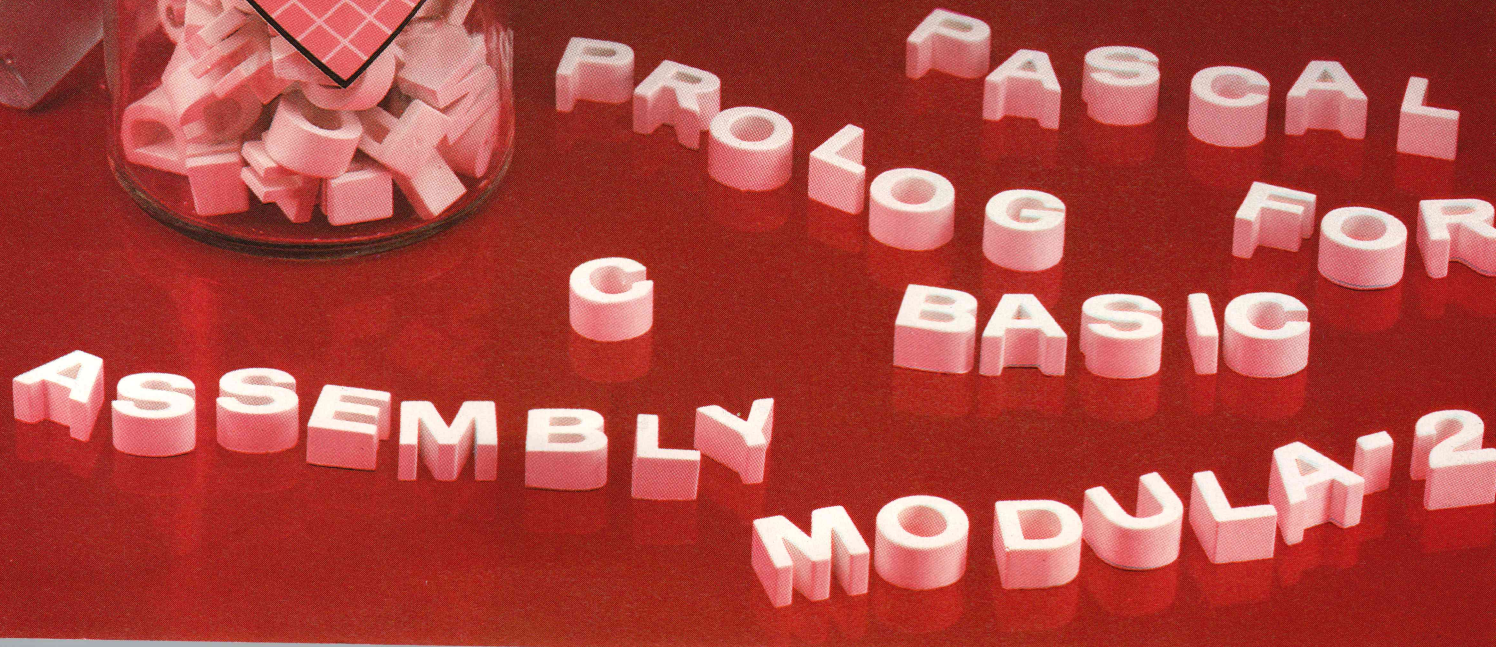
THE CURE FOR COMMON CODE



Are you getting the recommended monthly allowance of C, Assembly, Forth, Pascal, BASIC or Modula-2? Subscribe to *Dr. Dobb's Journal of Software Tools* and you won't catch any nasty bugs again!

Each month the Doctor brings you aid for ailing algorithms and the cure for common code. For the latest developments in software design and pages of code that will make you a more productive programmer, take the Dr. Dobb's prescription.

For more than a decade, the programming elite have known *Dr. Dobb's Journal* to be the foremost source of software tools. Subscribe now and get your monthly dose from the Doctor.



$5 SAVINGS

SUBSCRIBE AND SAVE!

Subscribe to

DR. DOBB'S JOURNAL OF SOFTWARE TOOLS

and save over $5—a 15% savings off the cover price!

☐ Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______ Exp. date ______

Signature ______

Name ______

Address ______

City ______ State ______ Zip ______

ONLY $29.97! YOU SAVE OVER $5.00

Savings based on a full one-year cover price of $35.40. Canada & Mexico add $10 for surface mail per year. All other countries add $27 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. 3454

COMMENTS & SUGGESTIONS

Dear Reader, **August 1987, #130**

Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take time to write a letter. This card provides you with an easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail.

—Ed

Which articles or departments did you enjoy the most this month? Why?

______

______

______

Comments or suggestions ______

______

______

______

Name: ______

Address: ______

______

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790 REDWOOD CITY, CA

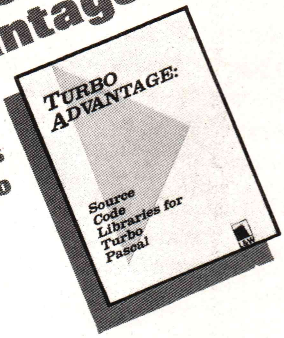
POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of
Software Tools
Box 3713
Escondidio, CA 92025-9843

Turbo Pascal Tools

Turbo Advantage

Source Code Libraries for Turbo Pascal



This library of more than 220 routines, complete with source code, sample programs and documentation will save you hours developing and optimizing your programs!

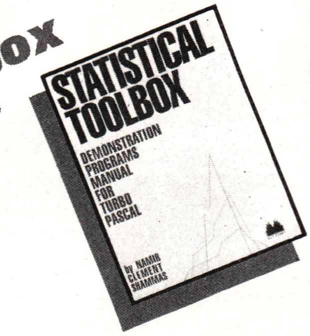
Routines are organized and documented under the following categories: bit manipulation, file management, MS-DOS support, sorting, string operations, arithmetic calculations, data compression, differential equations, Fourier analysis and synthesis, matrices and vectors, statistics, and much more! All source code is included.

A detailed manual includes a description of the routine, an explanation of the methods used, the calling sequence, and a simple example. For MS/PC-DOS systems.

Turbo Advantage Item #070 $49.95

Stat Toolbox

for Turbo Pascal



Two statistical packages in one!

A library disk and reference manual
Use these powerful statistical routines to build your applications. Routines include: • statistical distribution functions • random-number generation • basic descriptive statistics • parametric and non-parametric statistical testing • bivariate linear regression, multiple and polynomial regression.

A demonstration disk and manual
This package incorporates the library of routines into a fully functioning statistical program. Two data management programs are included to facilitate the storage and maintenance of data.

Full source code is included. (For IBM PC's and compatibles. Turbo Pascal version 2.0 or later, and PC-DOS 2.0 or later are required.)

STAT Toolbox Item #050 $69.95

Turbo Advantage Display

Form Generator for Turbo Pascal

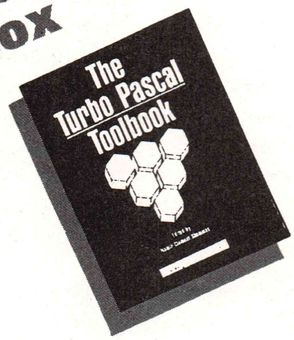
TURBO Display includes a menu driven form processor, 30 Turbo Pascal procedures and functions to facilitate linking created forms to your program, and full source code and documentation. For MS-DOS systems.

Some of the *TURBO Advantage: Source Code Libraries for Turbo Pascal* routines are necessary to compile *TURBO Display.*

TURBO Display Item #072 $69.95

Turbo Pascal Toolbox

Edited by Namir Clement Shammas



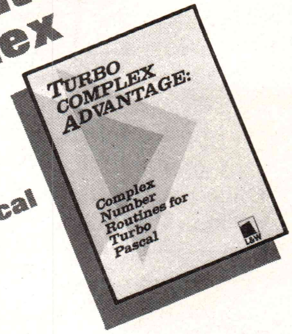
You'll find:
- an extensive library of low-level routines
- external sorting and searching tools, presenting a new database routine that combines the best features of the B-tree, B+ and B++ trees
- window management, to help you create, sort and overlay windows
- artificial intelligence techniques
- mathematical expression parsers, offering two routines that convert mathematical expressions into RPN tokens
- a smart statistical regression model that searches for the best regression model to represent a given set of data.

All routine libraries and sample programs are on disk for MS-DOS systems, and over 800K of Turbo Pascal source code is included!

Turbo Pascal Toolbook Item #080 $25.95
Turbo Pascal Toolbook with disk Item #081 $45.95

Turbo Advantage Complex

Complex Number Routines for Turbo Pascal



TURBO Complex provides procedures for performing all the arithmetic operations and necessary real functions with complex numbers. Each procedure is based on predefined constants and types. By using these declarations the size of arrays are easily adapted. Each type declaration is a record with both a real and imaginary part. Use these procedures to build more sophisticated functions in your own programs.

TURBO Complex also demonstrates the usage of these procedures in routines for vector and matrix calculation with complex numbers and variables; simultaneous Fourier transforms; calculations of convolution and correlation functions; low-pass, high-pass, band- pass and band-rejection digital filters; and solving linear boundary-value problems.

Source code and documentation is included. For MS-DOS systems. Some of the *TURBO Complex* routines are most effectively used with routines contained in *TURBO Advantage.*

TURBO Complex Item #071 $89.95

TO ORDER: Return this coupon with your payment to: M&T Books, 501 Glaveston Dr., Redwood City, CA 94063. Or, Call TOLL-FREE 800-533-4372 Mon-Fri 8a.m.-5p.m. In CA call 800-356-2002

YES! Please send me

- ☐ **TURBO Advantage** # 070 $49.95 ______
- ☐ **TURBO Advantage Display** # 072 $69.95 ______
- ☐ **TURBO Advantage Complex** # 071 $89.95 ______
- ☐ **STAT Toolbox** # 050 $69.95 ______
- ☐ **Turbo Pascal Toolbox** # 080 $25.95 ______
- ☐ **with disk** # 081 $45.95 ______

Subtotal ______
CA residents add salestax ____% ______
Add $2.25 per item for shipping ______
TOTAL ______

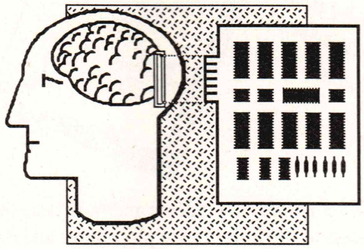
Name ______
Address ______
City ______
State ______ Zip ______

☐ Check enclosed. Make payable to M&T Publishing.
Please charge my ☐ VISA ☐ M/C ☐ AMEX

Card no. ______
Expiration Date ______
Signature ______

3128 A

LOOPS



This month I'll continue my evaluation of the Xerox 1186 LISP machine with an in-depth discussion of LOOPS, the object-oriented AI programming environment that runs on this machine. LOOPS was developed at Xerox PARC, and the first version was released in 1983. The principal designers of the original LOOPS system were Dan Bobrow and Mark Stefik. LOOPS has just recently been released as a commercial product and is an important addition to available expert system tools and AI development environments.

Before going into some of the details of LOOPS, I would like first to describe just what sort of an AI programming environment this is and what its overall significance might be. But first, I should remove one source of confusion—LOOPS is not the same thing as CommonLOOPS. The latter is a low-level, object-oriented extension to Common LISP whereas LOOPS is a high-level, AI language that already has most of the facilities needed for developing advanced AI applications.

As with most AI systems, LOOPS supports rule-based programming. What makes LOOPS unique, however, is its complete implementation of

by Ernest R. Tello

an object-oriented programming environment. The language contains just about all that was valuable and important in Smalltalk as well as much else besides. LOOPS was the tool used to create the PRIDE expert system developed by Sanjay Mittal, which I described in my first column (see *DDJ*, February 1987). It is already apparent that some new paradigms for expert system development have emerged as a result of various projects using LOOPS.

One of the central ideas in the design of the LOOPS environment was to provide an AI programming system that would support a multiple-paradigm framework. The current system supports four main programming paradigms: the object-oriented paradigm, the rule-based paradigm, the access-oriented paradigm, and the normal procedural paradigm.

Classes and Instances

As with all object-oriented programming systems, LOOPS provides for building hierarchies of classes and instances of those classes. Let's first look at the simple syntax used for accessing objects. The way you would reference a user-defined class called *Partnership*, or any other class, would be:

```
($ Partnership)
```

The dollar sign means that the object pointer to the *Partnership* structure is to be referenced. All references to objects in LOOPS use this convention of preceding the name of the object with the dollar-sign character.

Another syntax convention used in LOOPS is the back-arrow character, which I represent as <–. This character is accessed on the standard keyboard of the 1186 with the underscore key. The <– character translates roughly as "send the message" and corresponds to *message* in Flavors or *send* in SCOOPS (the object-oriented extension to PC Scheme). So, the LOOPS expression:

```
(<–($ Partnership) New 'OurVenture)
```

would send the message *New* to the *Partnership* class to create a new instance of itself called *OurVenture.*

Much of the activity in developing LOOPS applications involves the use of its rich variety of window- and menu-based tools, such as browsers and editors.

LOOPS Browsers

Software, as most programmers realize, is developed in layers, or shells, of functionality. All the major advances in software engineering coexist in some form, like different layers of an onion or rings of the trunk of a tree. LOOPS and the 1186 are both fine examples of this organic evolution in the AI field. The high-level tools that are provided with LOOPS offer an AI development environment that, in effect, takes software development to its next level.

One of the most useful and spectacular facilities in LOOPS is the visually oriented graphics class browser called the Lattice Browser. This facility has a main window that displays the class hierarchy with graphics lines depicting the lines of inheritance between classes. Many facilities for editing objects are available for use by interacting directly with this display. The main menu for the Lattice Browser facility looks like that in Example 1, page 131.

The *PrintSummary* command prints a full description of the selected class, including all its local variables and methods, in the Exec window. For example, selecting *ActiveValue* and using the *PrintSummary* command gives the display shown in Example 2, below. The *PrintSummary* operation has the convenient feature that the custom methods for classes are shown in bold type whereas the inherited classes are shown in normal type.

The *WhereIs* command is also convenient. If you need to know the class in which a particular method is first defined, all you have to do is choose this option, wait for a window with a list of all the methods in the system, and select one. Almost immediately, the name of the class involved on the Lattice Browser network display will blink on and off.

Developing applications in LOOPS involves a combination of writing code in the editor and accessing a large number of convenient facilities in the mouse-oriented window and menu environment. One convenient way of developing object classes is simply to enter an empty class in the Exec window and use the interactive facilities to flesh out the class definition. For example:

```
(DefineClass 'Partnership)
```

Once you have entered a class in this way, you can then access it with the class browser. To do so, you call up the main menu, select the *Browse Class* command, and then type in the name of the root class at the prompt. Another way you can tell LOOPS that you want to edit the methods of a class is by accessing them through the general Lattice Browser. The same menus are available there as under an individual class browser window. For me, the Lattice Browser and related classes form the heart of the LOOPS user interface.

Methods

In object-oriented systems, methods are the private procedures or functions known only to objects of a given class and its descendants. LOOPS has six different categories of methods—the class methods and object methods found in Smalltalk and other object-oriented systems and the Internal, Public, Masterscope, and Any method categories. Internal methods are the low-level system methods that implement LOOPS itself. They can be used by programmers who know what they are doing but are not intended for use as library methods to be specialized. Public methods are all those either provided with the system or developed by the user that are intended to be specialized for various purposes. Besides these, there are also special Masterscope methods that are local only to a particular application and can be used only when it has been invoked. Any methods are all those that have not been declared to be one of the other types.

Methods in LOOPS follow the syntax:

```
(METHOD ((ClassName Selector) self
        ARG1 . . . ARGn) . . . body)
```

Selector is the name of the method that, when sent to the appropriate object, succeeds in invoking it. The *self* argument is a dummy term that stands for the class to which the message will be sent. For example, the *Destroy* method, which is implemented for the *Object* class, is written:

```
(Method ((Object Destroy)
   self
   (<-(Class self)
       DestroyInstance self))
```

It takes no arguments other than *self* but is written so that it can be inherited by subsequent classes but still always destroy the proper class when called. The expression *(Class self)* assures that the message will be sent to the class of the object. From there, it simply calls on the *DestroyInstance* method. This may seem metaphysical, but practically speaking it is important to be able to uncreate objects to provide memory for creating other new ones.

Methods are created in LOOPS using *DefineMethod*. This function has the form:

```
(DefineMethod class selector
args OrFn expr file methodType)
```

The way you would usually go about defining a new method is to click the middle mouse button on the class in the Lattice Browser and then select the *Add* command from the menu and *AddMethod* from its submenu. At that point a prompt panel opens with the message:

```
Type the selector for the new method: >
```

You then enter the method's calling name. For example, let's say you enter the name *NewMethod*. At that point a window of SEdit opens with the following template already loaded in it:

```
(Method ((Object NewMethod)
        self

        (SubclassResponsibility))
```

```
 ________________________________
:       PrintSummary          > :
:     Doc (ClassDoc)          > :
:  WhereIs (WhereIsMethod)    > :
:    DeleteFromBrowser        > :
:       SubBrowser              :
:       TypeInName              :
:_______________________________:
```

Example 1: *The Lattice Browser's main menu*

```
#.($ ActiveValue)
Supers
Object
IVs

CVs

Methods
AVPrintSource AddActiveValue CopyActiveValue DeleteActiveValue
DeleteNestedActiveValue GetWrappedValue
GetWrappedValueOnly HasAV? NestActiveValue PutWrappedValue
PutWrappedValueOnly ReplaceActiveValue WrapOutside?
WrappingPrecedence
```

Example 2: *Display resulting from selecting* ActiveValue *and using* PrintSummary *in the Lattice Browser*

(continued from page 131)

This template is purely for convenience when it applies. If you like, you can delete any part of it or even all of it and begin with a clear editing window.

On the whole, the template is usually useful. To say that this is a huge workspace with vast resources that it takes substantial time to master would run the risk of understating the case.

Active Values

In LOOPS, an active value is an object that sends messages as a side effect of attempts to read or write to the instance variable of another object. This facility is often useful in visually oriented interfaces, debugging, initializing variables, and defining dependency relationships between variables. The *ActiveValue* class is a direct subclass of the *Object* class. It is an abstract class, however. Instances are not made of it but of its subclasses.

An interesting example of a practical use of active values is in designing a window that always remains square even when resized by the user. The way you do this is to create an active value that tracks the width variable for an instance of the *SquareWindow* class and automatically sets the height variable equal to it.

One of the built-in uses of active values in LOOPS is for dynamic monitoring of the state of objects. *Gauges* is a LOOPS library application that contains a variety of display classes that allow you to attach the values of critical variables to graphic displays. These displays depict various types of gauges and meters that provide for visual inspection of the instance variables of instantiated objects. Whenever the value of an attached variable changes, the gauge or meter is immediately modified to indicate the new value.

The *Gauges* class has two main subclasses—*LCD* and *Instrument*. *LCD* in turn has the two main subclasses—*Digiscale* and *Digimeter*. *Instrument*, on the other hand, has three main branches—*VerticalScale*, *RoundScale*, and *HorizontalScale*. *Meters* and *Dials* are specializations of the *RoundScale* class. All in all, these *Gauge* subclasses provide for just about any style of visual gauge or meter that might be needed, ranging from needle gauges to thermometers.

To use *Gauges*, all that is really involved is to create an instance of one of the *Gauge* classes and provide it with values for the necessary parameters by sending it the appropriate messages. To get a gauge to be visible on the screen, you would first create an instance by saying:

```
(<-$Dial New 'DialOne)
```

which creates an instance of the *Dial* class called *DialOne*. You would then send it the *Update* message:

```
(<-$DialOne Update)
```

If you need to set the value of the dial to an initial value, you can send the *Set* message:

```
(<-$DialOne Set 100)
```

The message that assigns a meter or dial to a given variable is the *Attach* message, which is also simple to execute. If you wanted to assign your dial as an indicator of the amount of

fuel left in a rocket in a simulation of a space vehicle mission, you could do so easily by saying:

```
(<-$DialOne Attach $TitanIV_547
             'FuelRemaining)
```

This *Gauges* application in LOOPS is similar to the *ActiveImages* facility in the KEE tool from IntelliCorp.

Mixins and Multiple Inheritance

LOOPS provides full support for multiple inheritance, which means that a class can be defined as a subclass of more than one superclass. Another way of saying this is that multiple superclasses can be selected as mixins for a new class.

The basic rule that inheritance follows in LOOPS can be stated succinctly as "left to right, up to joins." What this means is that, if a message M is sent to class Z and that method is not directly implemented in Z, then a search takes place up the class lattice for method M among the immediate superclasses of Z, their superclasses, and so on. The order of search is left to right and "breadth first" in the sense that all the immediate superclasses are searched first before any of their superclasses and so forth.

Rules

LOOPS has an original approach to using rules. Rules are always organized into definite rulesets, which can have various different kinds of control structure to evaluate them. A ruleset is always associated with some particular LOOPS object that provides the workspace for the rules. You can invoke rulesets in several different ways. In the object-oriented paradigm, you invoke them by sending a method to the object that contains them. In the access-oriented paradigm, you invoke them by using active values as a side effect of either reading or writing data in object properties. You can even write individual rules that invoke other rulesets, and you can also invoke rulesets from any LISP program.

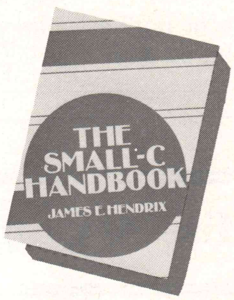
There are six main control structures for rule processing in LOOPS: *Do1*, *DoAll*, *While1*, *WhileAll*, *For1*, and *ForAll*. If you use the *DoAll* control structure, rule processing begins with the first rule of the ruleset and executes each and every rule that is satisfied. With the *Do1* control structure, only the first rule whose conditions are satisfied is executed. If no rule fires, the ruleset returns a value of NIL.

The *While1* control structure is a cyclic version of *Do1*. With this control regime, a *while* condition is specified. If the condition is satisfied, the first rule whose condition is satisfied is executed, as with the *Do1* construct. The difference is that if the *while* condition is still satisfied after that, the process is repeated until the condition no longer holds or until a *Stop* instruction is encountered. Similarly, the *WhileAll* construct is the cyclic version of *DoAll*. If the condition is satisfied, all the rules are tried and as many executed as can fire, and this is repeated until either the *while* condition fails or *Stop* is encountered.

The *For1* construct is another cyclic version of *Do1*. Instead of a *while* condition, this type of control structure has an iteration condition. The processing of rules occurs as with *Do1*, but the process reiterates over a range of values until the limit value is reached. A similar control regime occurs with the use of the *ForAll* construct, except that here the behavior

DR. DOBB'S C TOOLBOX

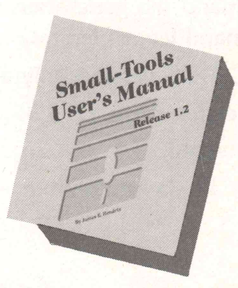
Small-C Handbook & Small-C Compiler



This compiler and handbook provide everything you need for learning how compilers are constructed, and for learning C at its most fundamental level. You'll find a discussion of assembly language concepts and program translation tools, and of how to generate a new version of the compiler. Full source code is included. Please specify MS/PC-DOS or CP/M: Kaypro, Osborne, Apple, Zenith Z-100 DS/DD, 8" SS/SD.

CP/M Compiler & Handbook	Item #006B	$37.90
MS/PC-DOS Compiler & Handbook	Item #006C	$42.90

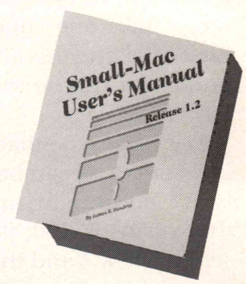
Small-Tools: Programs for Text Processing



This package of Small C programs performs specific, modular operations on text files. It is supplied as source code with full documentation. With the Small-C Compiler, you can select and adapt these tools to meet your needs. Please specify MS/PC-DOS or CP/M: Kaypro, Osborne, Apple, Zenith Z-100 DS/DD, 8" SS/SD.

Small-Tools	Item #010A	$29.95

Small-Mac: An Assembler for Small-C



This package includes: a simplified macro facility, C language expression operators, object file visibility, descriptive error message and an externally defined instruction table. Source code and documentation is included. CP/M only. Please specify: Kaypro, Osborne, Apple, Zenith Z-100 DS/DD, 8" SS/SD.

Small-Mac	Item #012A	$29.95

Special Packages

CP/M C Package
Save Over $27!

Receive: Dr. Dobb's Toobook of C, The Small-C Handbook and Small-C Compiler, Small-Mac Assembler, and Small-Tools Text Processing Programs. Only $99.95!

CP/M Package	Item #005A	$99.95

MS/PC-DOS C Package
Save $22

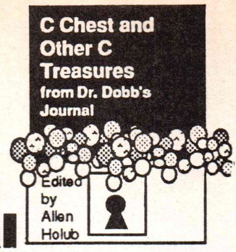
Receive: Dr. Dobb's Toolbook of C, The Small-C Handbook and MS/PC-DOS Addendum, Small-C Compiler, Small-Tools Texts Processing Programs and Small Windows. Only $109.95!
Specify Microsoft C Version 4.0, Small C or Lattice C compiler.

MS/PC-DOS Package	Item #005W	$109.95

C Disk Formats
Please indicate MS/PC-DOS or CP/M. For CP/M specify: Apple, Osborne, Kaypro, Zenith Z-100 DS/DD, 8" SS/SD.
For Small-Windows, specify Small-C, Microsoft C Version 4.0 or Lattice C compiler.

NEW

C Chest and Other C Treasures from Dr. Dobb's Journal



This comprehensive anthology contains the popular "C Chest" columns from *Dr. Dobb's*, along with the lively philosophical and practical discussions they inspired. You'll also find other information-packed articles by C experts.

Topics covered include: pipes, wild-card expansion, and quoted arguments; sorting routines; command-line processing; queues and bit maps; utilities such as *ls*, *make*, and *more*; expression parsing; hypenation; IBM cursor control and an Fget that edits; redirection; accessing IBM video display memory; trees; an AVL tree database package; directory traversal; sets; shrinking. EXE file images; hashing, expressions, and Roman numerals; and statistical applications of digital low-pass filters.

Other C treasures include: a variable metric minimizer; Christensen protocols; Fgrep; a peephole optimizer; curve fitting with cubic splines; and the CompuServe B protocol.

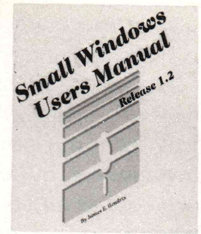
All subroutines and programs are written in C and are available on disk with full source code. MS-DOS format.

Book & MS-DOS Disk	Item #49-6	$39.95
Book only	Item #40-2	$24.95

To Order: Return this order form with your payment to **M & T Books, 501 Galveston Drive, Redwood City, CA 94063.**
Or, call TOLL-FREE 800-533-4372 Mon–Fri 8AM-5PM.
(In CA call 800-356-2002)

NOW FOR MICROSOFT C, SMALL C, AND LATTICE C COMPILERS

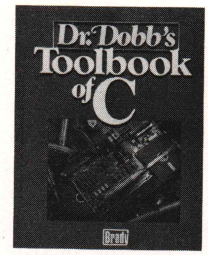
Small-Windows: A Library of Windowing Functions for the C Language



Small-Windows is a complete windowing library for C. The package contains: 18 video functions written in assembly language; 7 menu functions that support both static and pop-up menus; and 41 window functions, including functions to clean, frame, move, hide, show, scroll, push, and pop windows. A file directory facility illustrates the use of the window menu functions and provides file selection, renaming, and deletion capability. Two test programs are also included. For PC/MS-DOS systems, Microsoft C Version 4.0, Small-C, and Lattice C compilers. Documentation and full source code is included.

Small-Windows	Item #35-6	$29.95

Dr. Dobb's Toolbook of C



This authoritative reference contains over 700 pages, including the best C article from Dr. Dobb's Journal along with new material. You'll find hundreds of pages of valuable C source code, including a complete compiler, an assembler, and text processing programs.

Toolbook of C	Item #005	$29.95

ORDER FORM

M&T PUBLISHING, INC.
501 GALVESTON DRIVE
REDWOOD CITY, CA 94063

C Disk Formats

Please indicate MS/PC-DOS or CP/M.
For CP/M specify: Apple, Osborne, Kaypro, Zenith Z-100 DS/DD, 8" SS/SD.

For Small-Windows, specify Small-C, Microsoft C Version 4.0, or Lattice C compiler.

☐ Check Enclosed. Make Payable to M&T Publishing Inc.

Charge my ☐ VISA ☐ M/C ☐ Amer. Exp.

Card # ____________ Exp.Date ________

Signature ____________________

3130C

Name ____________________

Address ____________________

City ____________ State ______ Zip ______

Item #	Description	Price

Subtotal ________

CA residents add sales tax ______ % ________

Add $2.25 per item for shipping ________

TOTAL ________

(continued from page 133)

resembles *DoAll*—as many rules as can be satisfied are executed.

One of the main ideas behind the design of the LOOPS rule-oriented programming approach was to allow control information to be factored out as much as possible. This is, of course, a worthwhile idea because it means that the knowledge is kept separate from the control structure mechanisms. One of the advantages of rule-based programming is just this separation of content from control. It allows the modular addition of rules so that a production system keeps running from the time the first rules are entered until it is completed without rewriting the inferencing code. Some AI languages, such as OPS5, encourage the writing of numerous rules whose function is control of knowledge processing, which tends to neutralize the advantages of rule-based systems in separating knowledge and control. The LOOPS control structure declarations I have just outlined attempt to cope with this.

Another useful LOOPS rule construct is that of first/last rules, which are rules that can fire either before or after the main part of a ruleset is invoked. They are implemented by inserting an *{F}* or *{L}* in the *MetaDescription* field just prior to the rule proper. LOOPS also has an audit trail capability in its implementation of rules.

The rule syntax in LOOPS can best be illustrated by an example. Example 3, below, is an illustration from the LOOPS manual. In this example, the brace indicator *{1!}* indicates that the rules involved are "one-shot bang rules," or "try once" rules. The rules are tried only once, whether they pass or fail. Any declaration in curly braces before rules is called a metadescription in LOOPS. Another use of such metadescriptions is in the meta-assignment statements used for describing audit trails and rules. Audit trails provide a thorough facility for debugging and explaining why things happened the way they did.

Calls to custom InterLISP or Common LISP functions can be included in LOOPS rules in both premises and conclusions simply by enclosing them in parentheses. Similarly, LOOPS message-sending expressions can be nested in rules by enclosing them in parentheses and observing the back-arrow and dollar-sign conventions. Access to LOOPS instance variables in rules is done by using a colon (:) operator. So, for example:

$YourPartnership:industry = 'Law

is a rule declaration that assigns the value *Law* to the *industry* variable of the *YourPartnership* object. Similarly, access to class variables is provided with the double colon (::) operator.

```
RuleSetName: FillTub;
WorkSpace Class: WashingMachine;
Control Structure: WhileAll;
Temp Vars: waterLimit;
While Cond: T;

{1!} IF loadSetting = 'Small THEN waterLimit <- 10;
{1!} IF loadSetting = 'Medium THEN waterLimit <- 13.5;
{1!} IF loadSetting = 'Large THEN waterLimit <- 17;
{1!} IF loadSetting = 'ExtraLarge THEN waterLimit <- 20;


IF temperatureSetting = 'Hot
THEN HotWaterValve.Open ColdWaterValve.Close;


IF temperatureSetting = 'Warm
THEN HotWaterValve.Open ColdWaterValve.Open;

IF temperatureSetting = 'Hot
THEN ColdWaterValve.Open HotWaterValve.Close;

IF waterLevelSensor.Test >= waterLimit
THEN HotWaterValve.Close ColdWaterValve.Close;
     (Stop T)
```

Example 3: *An example of the LOOPS rule syntax from the* Xerox *LOOPS* Manual

Virtual Copies

One of the more interesting things in the LOOPS library is the provision for virtual copies of networks of instances. This is based on the insight that it can be useful to treat a group of instances as a unit that can be duplicated and tracked efficiently. The copies are virtual in two different ways. Only those properties of the instances that are modified are actually copied. Those that remain identical to the originals just "share" the values of the prototype. The copies are also virtual in the sense that only the specific instances that will be needed in processing are actually copied.

Any object that is to have a virtual copy must have a special class variable called *VirtualVS.* The value of this variable specifies which instance variables of the original object will be copied as opposed to being shared. The implementation of virtual copies is accomplished by two classes—*VirtualCopyMixin* and *VirtualCopyContext.* Virtual copies represent a kind of hybrid between classes and instances. They provide a medium-level mechanism whereby constructions such as perspectives and hypothetical reasoning can be implemented.

LOOPS Applications

With LOOPS it is possible to develop a wide variety of different AI applications. It is not simply a shell for the development of expert systems. Even in the case of expert systems, different paradigms for them have been developed using LOOPS that depart dramatically from the usual rule-based systems. The facilities I have been describing make it possible to develop knowledge-based systems that make little or no use of the rule-based paradigm. How, then, are such systems designed?

The PRIDE expert system that I discussed in my first column is one of the best examples of such a system to date. There has been much talk at Xerox about building an entirely new type of expert system shell paradigm based on the PRIDE application, just as the EMYCIN shell was derived from the MYCIN expert system application.

ARTIFICIAL INTELLIGENCE
(continued from page 136)

A few years ago, some interesting research on hierarchical planning in the LOOPS environment was conducted at Xerox PARC by the late Danny Berlin.

Some important work with the LOOPS system has also been conducted at Ohio State University under the direction of Professor B. Chandrasekaran. Professor Chandrasekaran is an advocate of what he calls "generic tasks" that operate as high-level building blocks in the development of knowledge-based AI applications. At this point, he feels that there are primarily six such generic tasks: hierarchical classification, hypothesis matching or assessment, knowledge-directed information passing, abductive assembly, hierarchical design by plan selection and assembly, and state abstraction. I will attempt to give only a brief explanation of these generic tasks here.

Hierarchical classification is perhaps the best-known type of problem in the expert systems category, a simple example of which is the well-known Animal game. It turns out that this problem of classification is at the heart of many diagnosis problems. Hypothesis matching is the process of determining the degree of fit of a collection of data points to a hypothesis, such as by estimating the probability or certainty that the hypothesis is true. Knowledge-directed information passing refers to the use of rules or frames to encode knowledge that directs a knowledge processing system to seek certain values under various conditions. Abductive assembly is another form of reasoning that assembles the best hypotheses for a given set of data by a method similar to the means-ends analysis used in the Dendral expert system. Hierarchical design by plan selection refers to a new type of task in expert systems technology—that of routine design. This new category of application is typified by two mechanical engineering expert systems—Aircyl and PRIDE. The last generic task is state abstraction, which involves a mechanism for predicting the consequences of actions by the use of qualitative simulation.

Amazing as AI workstations such as the Xerox 1186 may seem, some of this technology has already started rubbing off on powerful micros. Next month I will review a surprisingly powerful and cost-effective, object-oriented programming tool: Smalltalk/V for IBM PCs and compatibles.

Bibliography

Bylander, T.; and Mittal, S. "CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling." *AI Magazine*, vol. 7, no. 3 (Summer 1986).

Chandrasekaran, B. "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design." *IEEE Expert* (Fall 1986).

Mittal, S.; et al. "PRIDE: An Expert System for the Design of Paper Handling Systems." *Computer* (July 1986).

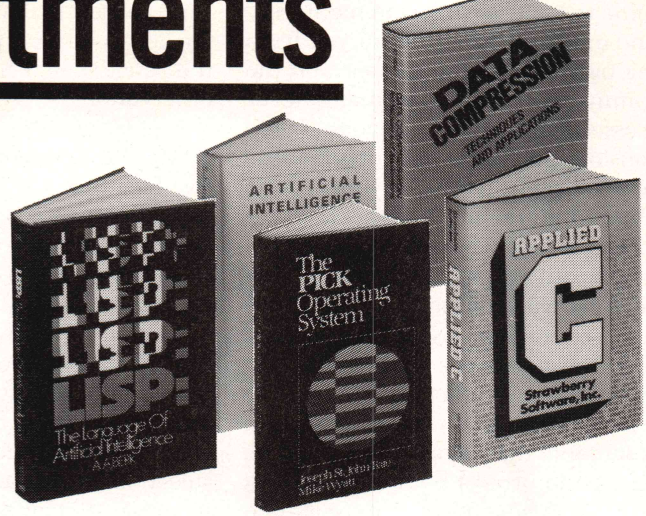
Mittal, S.; Bobrow, D.; and Kahn, K. "Virtual Copies, Between Classes and Instances." *ACM OOPSLA-86 Conference Proceedings*.

Xerox LOOPS Manual. Pasadena, Calif: Xerox Corp., 1987.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 8.**

Growth Investments



"An excellent book...which picks up where the introductory texts end."
—Choice

APPLIED C

By Stawberry Software, Inc.,
edited by Bonnie Derman

Let experts from Strawberry Software, Phoenix, Lattice, Greenleaf, and nearly two dozen of the companies that create C tools, libraries, and compilers help you get the most out of the C language. Their practical, problem solving approach covers writing program specifications...considerations of user interface design, style, portability, and modularity...data types, parsing, and indexing...and much more. **272 pp., illus., $37.95**

LISP

The Language of Artificial Intelligence
By A.A. Berk

Practical examples and careful explanations help you master the complexities of such concepts as recursion and data structure in this informative guide to LISP. BASIC is used as a reference point throughout. **168 pp., $28.95**

ARTIFICIAL INTELLIGENCE

An Applications-Oriented Approach
By Daniel Schustzer

Start putting AI to work for you *now* with this first guide to the current applications of expert systems, machine perception, and natural language processing. You'll find out how to use AI to solve a wide range of problems in business, industry, science, medicine, and defense, among other areas. **224 pp., illus., $39.95**

DATA COMPRESSION TECHNIQUES AND APPLICATIONS

By Thomas J. Lynch

Covers theoretical background, techniqes and applications of data compression in depth. You'll get details on system design considerations and examples of specific applications in speech, seismic data, electrocardiograms, x-rays, teleconferencing, television, videotex, and much more. **352 pp., illus., $46.95**

"Particularly useful to users wishing to develop their own systems."
—Computer World

THE PICK OPERATING SYSTEM

By Joseph St. John Bate *and* Mike Wyatt

With its relational data base and English Language interface, PICK is ideal for business applications. Here's everything you need to know about the PICK editor, the ACCESS and Advance ACCESS inquiry languages, the spooler, the PROC processor, and techniques for assuring system security. **160 pp., illus., $24.95 paper**

Available in December

COMPILER DESIGN AND CONSTRUCTION

Tools and Techniques (With C and Pascal)
Second Edition
By Arthur Pyster, Ph.D.

This state-of-the-art guide utilizes C language and the standard Pascal in a wealth of examples that demonstrate how to design, write, and implement compilers. Step-by-step instructions enable you to master all the techniques needed to write compilers at any level of complexity. *Uses UNIX™ tools.* **474 pp., illus., $35.95**

Mail today for RISK-FREE Examination!

VNR **Van Nostrand Reinhold,** Mail Order Dept.
P.O. Box 668, Florence, KY 41042-9979

YES! Please send me the book(s) checked below for 15 days' RISK-FREE Examination. At the end of that time I will send the full purchase price plus local sales tax and shipping/handling, or simply return the book(s) and OWE NOTHING.

___ 28217-6 Applied C $37.95
___ 20974-6 LISP $28.95
___ 28034-3 Artificial Intelligence $39.95
___ 534-03418-7 Data Compression Techniques and Applicaitions ... $46.95
___ 29276-7 The Pick Operating System .... $24.95 paper

Orders totalling over $100.00—Please enclose check, money order, purchase order, or credit card information. Payment in advance saves you shipping/handling costs and you'll receive a full refund if not completely satisfied. Local sales tax must be included.

☐ **SAVE MONEY!** Check here if enclosing payment with order and publisher pays shipping/handling. 15 day return/refund guarantee. Local sales tax must be included.

Name____________________

Address____________________
(No shipment to P.O. Box address without prepayment)

City____________________

State____________ Zip________

☐ **VISA** ☐**MasterCard** ☐**American Express**—Please charge my credit card.

Card #________________ Exp._____/_____

Signature____________________

Offer good in USA and territorial possessions only and subject to credit department approval. Prices subject to change and slightly higher in Canada

D 8434

VNR **VAN NOSTRAND REINHOLD**

CIRCLE 256 ON READER SERVICE CARD

LETTERS

(continued from page 12)

at fault for not making it clearer that the words defined by *VECTOR:* and the word *DO-OPTION* were conceived as getting their arguments from some routine (such as the menu routine mentioned in the text), not from the user entering a number and the command. I must admit that it never crossed my mind that anyone would consider giving the user unedited access to such words as these. The phrase Carl quotes, *4 DO-OPTION*, was intended as a warning, not as an example. (Although Carl points out that you can predict in general an unsatisfying result from using the phrase, I think he will agree that the specific effects are unpredictable.)

Carl and I again agree when he comments that programs should be written to protect both the program and the users. Carl's experience seems to be with command-driven programs; in those the command word must include the requisite edits and filters to qualify the input—and as he points out, a *CASE* statement normally includes implicit edits. (He refers to "the" *CASE* statement, but Forth provides no standard implementation. *CASE* statements are vendor dependent, which is why I did not use them in my article.)

My own programs are usually menu-based, with the edits built into the routine that presents the menu and collects the user's choice. Either approach—menu-driven or command-driven—can comfortably and reliably use execution vectors, provided the programmer made sure that the argument used to index into the vector is within range. Execution vectors are such a standard and useful tool in Forth programming that I think condemning their writers to Programmer's Hell is too severe. But, as Carl helpfully reminds us, execution vectors must be used with caution.

More Feedback

Dear *DDJ*,

With respect to Dan Farnsworth's letter and code example on page 12 of the April issue, I have the following observations:

1. Mr. Farnsworth's timings ignore the fact that *DBF* is faster when it branches than when it falls through. The correct times for his 68000 and 68008 loops are 5,672 and 3,784 cycles, respectively.
2. If you arrange the hardware so that each device register occupies either four successive even addresses, or four successive odd addresses, you can take advantage of the *MOVEP* instruction to produce a routine [see Example 1, below] that is 4 bytes longer than Mr. Farnsworth's 68008 loop routine but that takes 2,980 cycles on the 68000, compared to 3,784 for his on the 68008. The corresponding straight-line routine is 256 bytes longer than its 68008 counterpart, but it takes 2,332 cycles, compared to 2,616 for the 68008.

As an attempt to demonstrate that the 68008 can outrun the 68000, Mr. Farnsworth's example fails. I had, however, been using a rule of thumb that an 8-MHz 68008 has the throughput of a 4-MHz 68000 (except for multiplication and division, of course). It is clear that, at least in some special cases, a 68008 at 8 MHz can keep up with a 68000 at 6 MHz or better.

(Of course, in most cases, wait states would slow down any of those routines. To run with no wait states in an 8-MHz system, the peripheral controller would have to deliver a byte every 500 ns—an instantaneous rate of 2 megabytes per second. SCSI with handshake cannot do better than 1 byte every 667 ns, or 1.5 megabytes/second. Anyone who is prepared to spend the bucks for a synchronous SCSI channel with 2 megabytes/second or better throughput is likely to use a 68020 rather than a 68008 or 68000.)

Christopher T. Jewell
3900 Moorpark Ave.
San Jose, CA 95117

```
        MOVEQ    #63,D1         ;                  4
        LEA      DCA,A0         ;                 12
LOOP    MOVEP.L  0(A0),D0       ;24 * 64        1536
        MOVE.L   D0,(A1)+       ;12 * 64         768
        DBF      D1.LOOP        ;10 * 63 + 14    644
        RTS
;                16
                                                 ---
;                               Total time      2980 cycles
```

Example 1: *Alternative to Dan Farnsworths' 68008 loop routine*

Buttons and Gadgets

Dear *DDJ*,

Thank you for publishing Jan L. Harrington's article on Macintosh and Amiga interface programming (January 1987). It provided me with a good starting point for assembly-language programming on the Amiga.

Please note, however, that some typographical errors appeared in the Amiga program listing (Listing Three) [see Table 1, page 142]. The first four typos generate assembler errors, and the last prevents exiting when you select "Quit" on the Project menu.

James P. Hastey, Jr.
c/o PPCoN, Dept. 90
P.O. Box 220
N-4056 Tananger
Norway

Dear *DDJ*,

This is a follow-up to my previous letter on typographical errors in the Amiga program listing accompanying Jan Harrington's article.

My exuberance in actually getting the program to run caused me to overlook two shortcomings: the Workbench memory counter reported 176 bytes (cumulative) less of free memory each time the program ran plus the system crashed after five to eight successive runs.

Closer examination revealed that the program needed to perform a little more housecleaning before exiting. Perhaps the omission was intentional, with the listings accompanying the article intended only as examples of the difference in the two user interfaces. As such, they served the purpose well. For the Amiga listing (Listing Three) to be a more practical example, a few additions need to be made.

This is not a complaint. I thoroughly enjoyed reading Jan Harrington's

(continued from page 140)

article and learning a bit about the Amiga Intuition interface in assembly language. I would like to point out that I am not a programmer by profession but have worked with 6502 and 8088 assembly language as a hobby.

The following additions I propose are based solely upon recommendations found in the *Amiga ROM Kernel Reference Manual: Exec*, the *Amiga ROM Kernel Reference Manual: Libraries and Devices*, and the *Amiga Intuition Reference Manual*. On the surface they appear to cure the problems I've described. Please be aware that as a relative novice I may have overlooked something.

Add the following to the list of *xlibs* in lines 23–41:

```
xlib     FreeSignal
xlib     FreeMem
xlib     RemPort
xlib     CloseLibrary
xlib     ClearMenuStrip
```

Line	Appears in Listing as	Should Read
5	CALLIB_LVO\1	CALLLIB_LVO\1
23	(This instruction is missing	
to	from the series of xlibs in	
	lines	xlib OpenScreen
41	23 to 41.)	
62	move.l #0,ns_Fonts(A0)	move.l #0,ns_Font(A0)
122	lea ProjItem,A1	lea ProgItem1,A1
226	and #%0000000000111111,DO	and#%0000000000111111,D1

Table 1: *Corrections to Listing Three of Jan Harrington's article (January 1987)*

```
move.l   ReadMsg,A1
move     #IOSTD_SIZE,D0
move.l   _AbsExecBase,A6
callsys  FreeMem                  ;free up read io block memory

move.l   ReadPort,A1
move.l   _AbsExecBase,A6
callsys  RemPort                  ;remove read port from system

move.l   ReadPort,A1
move     #MP_SIZE,D0
move.l   _AbsExecBase,A6
callsys  FreeMem                  ;free up read port memory

move.l   ReadPort,A4
clr.l    D0
move.b   MP_SIGBIT(A4),D0
move.l   _AbsExecBase,A6
callsys  FreeSignal               ;free up read port signal bit

move.l   WriteMsg,A1
move     #IOSTD_SIZE,D0
move.l   _AbsExecBase,A6
callsys  FreeMem                  ;free up write io block memory

move.l   WritePort,A1
move     #MP_SIZE,D0
move.l   _AbsExecBase,A6
callsys  FreeMem                  ;free up write port memory

move.l   WritePort,A4
clr.l    D0
move.b   MP_SIGBIT(A4),D0
move.l   _AbsExecBase,A6
callsys  FreeSignal               ;free up write port signal bit

move.l   WindowPtr,A0
move.l   IntBase,A6
callsys  ClearMenuStrip           ;clear menu strip
```

Example 2: *Insert for subroutine Close And Quit*

Insert the code in Example 2, below, between lines 233 and 234 in the subroutine *CloseAndQuit*, and insert the following between lines 239 and 240 of the same routine:

```
move.l   IntBase,A1
move.l   _AbsExecBase,A6
callsys  CloseLibrary       ;close the
                                library
```

James P. Hastey, Jr.
(address on previous letter)

Jan Harrington replies:

My thanks to James Hastey for finding the typographical errors in the Amiga listing. They arose primarily because the Amiga code, after being debugged on that machine, was keyed into the Mac from printed listings to prepare copy for typesetting (I suppose the smarter way would have been a direct machine-to-machine transfer). His housekeeping additions to the code are also well taken. The purpose of that program, however, was simply to demonstrate the Amiga user interface, not to produce a program that would actually do useful work. As it was getting very long, even with just the user interface code, I decided to keep it as simple as possible.

Macintosh programs, in general, do not require the same sort of cleanup as Amiga programs do. Under ordinary circumstances, the Mac's operating system performs the cleanup on its own; programmers needn't worry about it.

The Right Tool

Dear *DDJ*,

I refer to Mike Suman's Viewpoint in your February 1987 issue. I wholeheartedly agree with Mr. Suman's analysis of what is wrong with high-level languages. Though I am a high-level language user and have never written production programs in assembly language (except in computer school), I must admit that at times they make the life of the programmer difficult, if not clerical.

I differ with Mr. Suman's view with regard to the assembly-language version of Mr. Anderson's Modula-2 program (Code Example 3), however. I have never programmed

in Modula-2, but between two evils I still prefer Modula-2 for I find no thrill or excitement in writing strings of 1s and 0s as shown in the example. I feel that programmers who have been exposed to a wide variety of languages would agree with me in this case. We must bear in mind that the main aim of a high-level language is to unburden programmers from dealing with trivial things so they can concentrate on the main problem at hand.

This is why I feel that dBASE III languages are best when it comes to handling file and tabular problems. They provide the right ingredients for attacking trivial problems such as the one discussed. To dBASE programmers the benefit is obvious right away—ease of maintenance. I wish facilities such as this (table handling) would be included in the new languages flooding the market. Furthermore, we must not forget that every language is designed to suit a particular problem, and we should therefore use the right tool for the job. We should not use a tool for something for which it is not intended. We border on the unreasonable when we overstretch the validity of a thing.

Lito Cruz
3 Spring St.
Thomastown, Victoria
Australia 3074

Correction

Dear *DDJ*,

As the author of "An Extended IBM PC COM Port Driver" (June 1987), I wanted to get a head start on readers' complaining about bugs in Listing One presented with the article. The seventh line of code after the label *b000* needs to have *BUFSZ * 2* changed to *(BUFSZ * 2) – 2*. This bug causes COM1 operation to mess up COM2's buffer pointers and COM2 operation to destroy the ability to restore *int0B*'s vector upon termination. There's no problem at all if you only use COM1. The seventh line (I thought 7 was a lucky number) of code after the label *b230*, with the comment *indicate receiver enabled*, needs to be moved to be the fifth line after *b230*—that is, after the *jg . .* line. This bug causes DTR protocol to work only if XON/XOFF protocol is used also.

I apologize for any inconvenience and suggest that readers try excom, Version 2, available through *DDJ* or CompuServe. It has these bugs fixed, some substantial enhancements, and probably has some new and more exciting bugs.

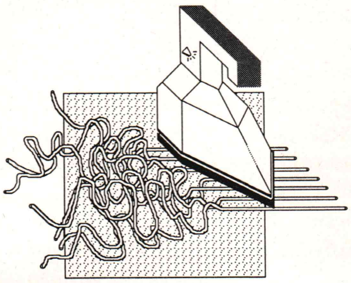
Also, the following code was deleted from the bottom of page 76:

```
                    init |= thebit;
}
/* set port number */
void
```

Tom Zimniewicz
2695 Pond Rd.
Lima, NY 14485

DDJ

THE STATE OF BASIC



The New Internal Coding Engines

In this issue we look at the decision-making and loop constructs implemented by the new BASICs. These contructs enable you to write more structured code and use fewer *GOTO*s. Most of these constructs are derived from features of other languages, notably Pascal. Does that mean that BASIC is being Pascal-ized? No! It simply indicates that a well-thought-out and more structured program is superior to an unplanned, *GOTO*-riddled program. Clarity and neatness of code really pays off when you come back later to update your program.

The new decision-making constructs have much to offer. First, the one-line *IF . . . THEN . . . ELSE* statement has been extended to spread over multiple lines. Moreover, *ELSEIF* clauses are now supported. This syntax for the *IF* statement is a much needed improvement. The ability for the *THEN* and *ELSE* clauses to contain a series of statements enables you to phase out *GOTO*s painlessly. Example 1, right, shows the general syntax of the extended *IF* statement.

The new BASICs have also added a *CASE* statement. QuickBASIC (Version 3.0), Turbo BASIC, and True BASIC have implemented *SELECT CASE* with features that outperform the *CASE* statements of Pascal and Modula-2. Example 2, right, demonstrates all the types of *CASE* statements in the new *CASE SELECT*. They are:

- single items to be compared with the selected variable
- a list of items, delimited by commas
- a range of values to be compared with the selected variable
- partial logical expressions

CASE statements can also contain a combination of these.

Looking at Example 2, notice that the first *CASE* statement compares the selected string *A$* with a single item. The second *CASE* contains a list of selected symbols. The following three *CASE* statements use the value range *(<first> to <last>)* to detect if the input character is lowercase, uppercase, or a digit. The following two *CASE* statements use the inequality operator to test if the character is a control character or an extended ASCII character. Finally, the *CASE ELSE* clause is an important catchall clause.

The new BASICs also include a new loop construct—namely, the *DO . . . LOOP*, a powerful and flexible loop that has the ability to use logical testing. The standard *FOR . . . NEXT* loop has been supported by adding an *EXIT FOR* statement to enable a clean exit from a *FOR* loop. Turbo BASIC also supports *EXIT* statements for the *WHILE* loop and *IF* and *CASE* statements.

```
IF <expression> THEN
   <sequence of statements>
ELSEIF <expression> THEN
   <sequence of statements>
ELSE
   <sequence of statements>
END IF
```

Example 1: *General syntax of extended* If *statement*

```
INPUT "Enter Character ";A$
IF A$ = "" THEN A$ = " "
A$ = LEFT$(A$,1)

SELECT CASE A$

   CASE "+"
      PRINT "Plus sign"
   CASE "!","@","#"
      PRINT "Special symbols"
   CASE "a" to "z"
      PRINT "Lower case"
   CASE "A" to "Z"
      PRINT "Upper case"
   CASE "0" to "9"
      PRINT "Digits"
   CASE is < CHR$(27)
      PRINT "Control character"
   CASE is > CHR$(127)
      PRINT "Extended ASCII set"
   CASE ELSE
      PRINT "Not classified by
                        this program
   END SELECT

   END
```

Example 2: *Short program to demonstrate SELECT CASE*

The bare bones *DO . . . LOOP* is an open loop that is exited from with an *EXIT LOOP/DO* statement. The *DO . . . LOOP EXIT* statement is embedded in the loop body. The *WHILE* and/or *UNTIL* clauses can be placed after the *DO* or *LOOP* keywords, as in:

```
DO [[WHILE ¦ UNTIL] <expression>]
<sequence of statements>
LOOP [[WHILE ¦ UNTIL] <expression>]
```

This creates an interesting combination of tests, especially because *WHILE/UNTIL* clauses can occur simultaneously after the *DO* and *LOOP* keywords. The powerful *DO . . . LOOP* can easily offer the equivalent of *WHILE . . . WEND* (in BASIC itself), *REPEAT . . . UNTIL* (in Pascal), and *DO . . . WHILE* (in C).

In addition, loops using *UNTIL <expression>* clauses can easily replace the equivalent *WHILE NOT <expression>*. Thus, the familiar file-reading loop:

```
OPEN 1,"I","DATA.DAT"
WHILE NOT EOF(1)
<file input operations>
WEND
CLOSE #1
```

can be replaced by:

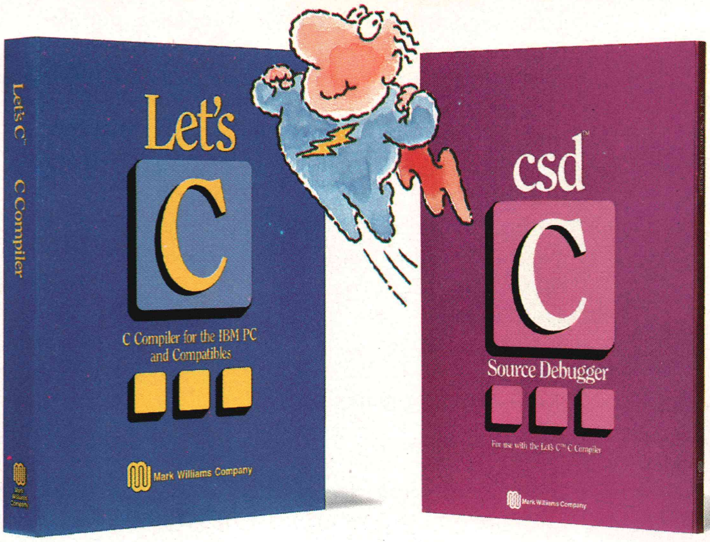
```
OPEN 1,"I","DATA.DAT"
DO UNTIL EOF(1)
<file input operations>
LOOP
CLOSE #1
```

This State of BASIC is the last in a series of introductory material on aspects of the new BASICs. Future issues of *DDJ* will contain articles that discuss, in an integral fashion, various aspects of programming with BASIC.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 9.**

PRESENTING THE DIFFERENCE BETWEEN FAST COMPILING AND FAST PROGRAMMING.



For compiling speed, you can't do better than Let's C. But to really speed up programming you can't do without the powerful source level debugger, *csd*.

If you want the power, portability and flexibility of C, start with the complete compiler, Let's C. For utilities, editor, compiling speed and fast, dense code, Let's C has it all.

But to get your programs up and running you need more. Because even the fastest compiler can't outrun bugs. You need the revolutionary C Source Debugger, *csd*.

CUT DEVELOPMENT TIME IN HALF WITH *csd*

csd lets you bypass the time consuming frustrations of debugging–like long dumps and clunky assembler. With *csd*, you actually debug in C. You learn faster because you watch your program run in C. You finish faster because *csd* combines the speed of a compiler with the interactive advantages of an interpreter. The end result? Development time is sliced in half.

LIMITED TIME OFFER FREE *csd* WITH LET'S C!

REVIEWERS ARE RAVING ABOUT LET'S C AND *csd*.

"Let's C is an inexpensive, high-quality programming package... with all the tools you will need to create applications."
–William G. Wong, *BYTE*, August 1986.

"The performance and documentation of the $75 Let's C compiler rival those of C compilers for the PC currently being sold for $500...highly recommended..."
–Marty Franz, *PC TECH JOURNAL*, August 1986.

"csd *is close to the ideal debugging environment...a definite aid to learning C and an indispensable tool for program development."*
–William G. Wong, *BYTE*, August 1986.

"This is a powerful and sophisticated debugger built on a well-designed, 'serious' compiler."
–Jonathon Sachs, *Micro/Systems Journal*, April, 1986

LET'S C AND *csd* FEATURES

NEW VERSION 4.0!

Let's C:

- Now compiles twice as fast
- Integrated edit-compile cycle: editor automatically points to errors
- Includes both small and large memory model
- Integrated environment or command line interface
- 8087 sensing and support
- Documentation features new lexicon format
- MS-DOS object compatible
- New make utility
- Fast compact code plus register variables
- Full Kernighan & Ritchie C and extensions
- Full UNIX compatibility and complete libraries
- Many powerful utilities including make, assembler, archiver, cc one-step compiling, egrep, pr, tail, wc
- MicroEMACS full screen editor with source included
- Supported by dozens of third party libraries
- For the IBM-PC and Compatibles
- Not copy protected

Sieve Benchmark
(Compile time in seconds)
Let's C: **2.8** (On 512K 6Mhz IBM-AT)
Turbo C: **3.89** (As advertised)

csd:

- Large and small memory model
- Debug in C source code, not assembler
- Monitor variables while tracing program
- Does not change program speed or size
- Provides separate source, evaluation, program and history windows
- On-line help screens
- Can interactively evaluate any C expression
- Can execute any C function in your program
- Trace back function
- Ability to set trace points
- Not copy protected

START TO FINISH, THERE'S NO BETTER ENVIRONMENT.

Get started with the right C compiler and you'll have everything you need for development–including source level debugging. On top of it all, Let's C and *csd* are today's best values in professional C programming tools. And most reliable: Mark Williams C compilers have been sold with DEC, Intel and Wang computers since 1981.

60 DAY MONEY BACK GUARANTEE

Mark Williams gives you a full 60 days to find out just how good Let's C and *csd* really are–or your money back.

So if you want more than a fast compiler–if you want your programs up and running fast, ask for Let's C and *csd*. You'll find them at your software dealer's, in the software department of your favorite bookstore, through the Express Program at over 5500 Radio Shacks or you can order now by calling **1-800-MWC-1700.*** *DDJ 8/87*

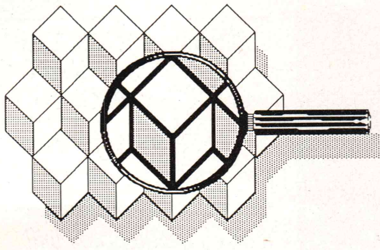
*In Illinois call, 1-312-472-6659.

Mark Williams Company
1430 West Wrightwood, Chicago, Illinois 60614

MARK WILLIAMS LET'S C AND *csd*. ONLY $75 EACH.

CIRCLE 102 ON READER SERVICE CARD

OF INTEREST



Spring Comdex in Atlanta is usually a software show, but this year, despite a conference program that focused on software issues, the exhibit floor contained a lot of hardware products. In terms of delivery, some of the PS/2 add-on hardware was softer than the 386 system software, of which there was also an abundance.

386 Computers

Wyse has announced a line of 286 machines and a 386 machine, all planned to track IBM and OS/2 but with some features that make the machines interesting to technical users. Each machine has front-panel controls and an LCD system-status display, disk caching, software emulation of the LIM *expanded memory spec*, and a disk reorganization utility to improve hard-disk access time. The 386 machine can support both the 80287 and the 80387 math coprocessors; it costs $4,999 for a 40-megabyte hard-disk model.

Wyse has cut prices on older models, for example, an AT-compatible now costs $1,999. Wyse tube subsidiary Amdek has also entered the personal computer market with its own line of 8088- through 80386-based machines. Reader Service No. 16.
Wyse Technology
3571 North First St.
San Jose, CA 95134
(408) 433-1000

TeleVideo has introduced four 386 computers, some with concurrent Unix and DOS, with prices starting at $3,995. One, TeleStar, comes with an 80387 coprocessor. Another, Telenix, uses Microport's V/386 Unix and DOS-Merge software, allowing Unix and DOS applications to run concurrently. TeleVideo thinks Unix/DOS integration's hour is nigh, citing 32-bit processor performance, convergence on System V for 32-bit systems, and good sales in key markets for Unix in the past few months. TeleVideo bought into Microport Systems, a company with a pivotal role in Unix/DOS integration, in March of this year. Reader Service No. 17.
TeleVideo
1170 Morse Ave.
P.O. Box 3568
Sunnyvale, CA 94088-3568
(408) 745-7760

In the continuing saga of **Unisys** reorganization, the company has regrouped a number of former Sperry and Burroughs Unix-based computers into one line running System V.2 from AT&T. Prices range from $14,000 to $500,000. Unisys has also announced a 386 workstation called the B38 for $4,835, or $5,635 with an 80287 coprocessor. Reader Service No. 18.
Unisys Corp.
P.O. Box 500
Blue Bell, PA 19424-0001
(313) 972-9515

Intelligent Data Systems, with heavy backing from Taiwan clone maker **Copam**, has entered the 386 market with a 16-MHz 386 machine with 1-megabyte RAM, 1.2-megabyte floppy drive, two serial ports, one parallel port, DOS 3.2, a 40-megabyte hard disk, and EGA graphics for $4,495. Reader Service No. 19.
Intelligent Data Systems
6319 East Alondra Blvd.
Paramount, CA 90723
(213) 633-5504

Contrary to popular belief, the leading mail-order manufacturer of personal computers is not an Austin-based company named **PC's Limited**. The company that has been doing business as PC's Limited was actually incorporated as **Dell Computer Corporation** in 1984, and it announced at Comdex that it would henceforth be using its real name, although existing products will continue to carry the PC's Limited logo. The change is undoubtedly because of a desire to get away from the low-budget mail-order image and to problems the name would cause as the company pushed into international markets. The fact that 22-year-old founder Mike Dell has acquired a degree of fame may have had something to do with it, too. The company continues to challenge the more pin-striped manufacturers on price and performance, has cut prices on its 286 machines, and is pushing price comparisons of its 386 machine with Compaq's and IBM's. The 386^{16} systems start at $4,499 for a 40-megabyte hard drive, monochrome system. Reader Service No. 20.
PC's Limited/Dell Computer Corp.
1611 Headway Cir., Bldg. 3
Austin, TX 78754
(512) 339-6800

SOTA Technology claims that you don't really need a 386 machine, citing benchmarks on which its 286 Mothercard 5.0 outperforms a Compaq Deskpro 386. The PC or compatible card comes with up to 4 megabytes on-board RAM and a 12.5-MHz 286 processor; a daughtercard can increase RAM to 12 megabytes and uses only one bus slot. An EPROM and battery-backed RAM hold the BIOS, making it eminently reconfigurable, so SOTA can pursue OS/2 compatibility. Reader Service No. 21.
SOTA Technology
657 North Pastoria Ave.
Sunnyvale, CA 94086
(408) 245-3366

386 System Software

The **THEOS** multiuser, multitasking operating system has been enhanced for the 386. Version 2.2 addresses up to 16 megabytes of real memory in protected virtual address mode, can read and write DOS files from/to a DOS partition, and supports the 80387 coprocessor. Reader Service No. 22.
THEOS
1777 Botelho Dr., Ste. 360
Walnut Creek, CA 94596-5022
(415) 935-1118

The Software Link is shipping PC-MOS/386, the multiuser, multitasking

(continued from page 146)

operating system for 386 machines that we discussed in July. Early reports are mixed; we'll say more when we know more. Reader Service No. 23.
The Software Link
3577 Parkway Ln.
Atlanta, GA 30092
(404) 448-5465

Quarterdeck has announced Version 2.0 of DesqView, its multitasking, multiwindow environment. Version 2.0 has VGE and MCGA graphics, supports Microsoft Windows-specific, Digital Research GEM-specific, and IBM Topview-specific applications; and EGA's 43-line and VGA's 50- and 60-line text modes. There's a full API capability so programmers can develop to the Deskview look and feel and a 386 memory manager à la Compaq's on the Deskpro 386. Suggested retail price is $129.95. Reader Service No. 24.
Quarterdeck Office Systems
150 Pico Blvd.
Santa Monica, CA 90405
(213) 392-9851

DesqView looks particularly impressive when one of its tasks is actually running on **Definicon's** DSI-780 68020/68881 card. The 780 line is just the latest of Definicon's 32-bit coprocessor products, which the Definicon folks have described in the August and September 1985 and July and August 1986 issues of *Byte*. They now claim that the 780 in an AT runs AutoDesk's AutoCad faster than any other system does, including the best unannounced 386 machine and the top-of-the-line Sun workstation. Reader Service No. 25.
Definicon Systems Inc.
1100 Business Center Cir.
Newbury Park, CA 91320
(805) 499-0652

OS/286 and OS/386 from **AI Architects** are extensions to DOS 3.x that permit programs to run in protected mode and address several megabytes of memory. They do not replace DOS; system calls still work and the operating system still looks like DOS to the user. AI Architects is the developer of Hummingboard, a PC card with up to 24 megabytes of on-card RAM and a 16-MHz or 20-MHz 386. The Hummingboard does not replace the 286 or 8086 CPU in the way that an accelerator card does but implements a smooth coprocessor design. Reader Service No. 26.
AI Architects Inc.
One Kendall Sq., Ste. 2200
Cambridge, MA 02139
(617) 577-8052

For people using 386 systems for multiuser purposes, **Arnet** has developed an add-on board called the Virtual Terminal Adapter that makes the 386 think that dumb terminals are video RAM. The board has four RS-232 ports. The problems the board solves have to do with running DOS applications, multitasking, and supporting local printers. Many DOS applications write directly to video RAM, so they don't work with dumb terminals. Multiuser operating systems generally do not support multitasking on dumb terminals because they are not memory-mapped, and multiuser systems usually cannot drive both the dumb terminal and a printer attached to it. Mapping the dumb terminals into video RAM and saving the 386 the associated housekeeping chores solves all these problems, Artek claims. The board's price is $1,500–2,000. Reader Service No. 27.
Arnet Corp.
476 Woodycrest Ave.
Nashville, TN 37210
(615) 254-0646.

PS/2 Add-Ons

Many board companies announced the launching of PS/2 product lines, but reading between the lines showed that often the only product near release was a board for the non-Micro Channel Model 30.

Orchid claims orchids for exhibiting the first memory board for the IBM PS/2 with a 2-megabyte Micro Channel board for $995. It's designed for Model 50/60 machines and will support both LIM Expanded Memory Spec and extended memory, which later will matter when Microsoft releases OS/2.

Orchid has also announced JetRAM, a 32-bit RAM board with 2 megabytes of extended memory for DOS or Unix users, and a graphics card compatible with PGA, CGA, EGA, MDA, and Hercules graphics. Reader Service No. 28.
Orchid Technology
45365 Northport Loop West
Fremont, CA 94538
(415) 683-0300

Tecmar is heavily committed to supporting PS/2 machines. Tecmar, you may recall, was quick to deliver a third-party hardware product for the IBM PC back in 1981, and it would like to be near the front of the PS/2 pack. Announced products include a memory/multifunction board with up to 2 megabytes and two serial ports. Reader Service No. 29.
Tecmar Inc.
6225 Cochran Rd.
Solon (Cleveland), OH 44139-3377
(216) 349-0600

Quadram announced several PS/2 line products, including a 2-megabyte memory board, 2-megabyte multifunction board, and I/O boards for Models 50 & 60. Only 512K- and 1-megabyte versions will be available at release this summer, using 256K SIMMs (Single In-line Memory Modules); the 2-megabyte version will use 1-megabyte SIMMs. Quadram also announced an 80287-based graphics board, offering thrice the clarity, 16 times the color selection, and 25 times the speed of EGA, now shipping; modes include 800 × 600 × 4. Reader Service No. 30.
Quadram
One Quad Wy.
Norcross, GA 30093-2919
(404) 923-6666

DDJ

The Hard Facts about Software Dealers

We're Programmer's Connection, your best one-stop source for quality programmer's development tools for IBM personal computers and compatibles. Here are some important facts you should know about us and other dealers in our industry.

FACT:

FREE Shipping. Shipping to U.S. customers is FREE via UPS Ground. If you want your order shipped via an express service, we'll only charge you the shipping carrier's standard rate with no special fees. Some dealers charge extra for shipping and then add rush charges for shipping via express services. Others may advertise "free" shipping but make up for it by charging extra handling fees.

FACT:

Credit Cards. We'll charge your credit card only when we actually ship your order. Some dealers would charge your credit card at the time you place your order. This could leave you waiting for your shipment for weeks or months while they use your money interest-free.

FACT:

Discounts. We discount all software products — even special order items. Every product in our advertised price list is shown with its list price and discounted price. We want you to know exactly how much you'll save on every product. We don't try to fool you by discounting some products and charging full retail for others.

FACT:

Consistent Prices. We extend the same current prices to every customer regardless of where they see our ad. Some dealers vary prices in different ads and then ask you to mention which one you saw. This technique allows them to charge you the highest prices possible.

FACT:

No Hidden Charges. The discount prices you see on the next two pages are all you pay. We don't charge extra for UPS Ground shipping, credit cards, COD orders, purchase orders, sales tax (except Ohio) or special handling (except for non-Canadian international orders).

FACT:

Guarantees. We offer FREE 30-day no-risk return guarantees and 30-day evaluation periods on most of our products. Some dealers have no return options while others often charge restocking fees of 15% or more.

FACT:

Quality Products. Our product line consists of hundreds of high quality software development tools specifically for IBM Personal Computers and compatibles. While some dealers try to carry every software product ever written, we carry only those that meet our very high standards for quality and value.

FACT:

Latest Versions. The products we carry are the latest versions and come with the same manufacturer's technical support as if buying direct. While some dealers may participate in the software gray market, we're authorized to sell every product we carry.

FACT:

Large Inventory. We have one of the largest inventories of programmer's development products in the industry. Most orders are shipped within 24 hours. And if we don't have a product in stock, we'll get it for you fast.

FACT:

Meticulous Packaging. We'll give your shipment the extra protection needed to reach you in the best possible condition. First we'll protect your products from moisture by wrapping them in plastic. Then we'll insulate your box with high quality bubble-wrapping instead of the messy styrofoam chips that many other dealers use. Finally, we'll double-tape your box for extra strength.

FACT:

Independence. Since we're not directly affiliated with any software publisher or developer, we can give you sound, unbiased advice. Unlike some other dealers who have a special interest in promoting only certain products, we'll give you an objective look at the products we carry.

FACT:

Noncommissioned Staff. Our courteous sales staff is always ready to help you. And if you aren't sure about your needs, our knowledgeable technical staff can give you sound, objective advice. Because they are noncommissioned, you won't be pressured into making a purchase.

As you can see, we're different from the other dealers in our industry. Our customers keep coming back because we consistently provide the highest quality service and the lowest prices. So call us today and experience the differences for yourself.

Turn the page for our product list and ordering information.

ai - expert systems

	List	Ours
1st-CLASS *by Programs in Motion*	495	399
AutoIntelligence *by IntelligenceWare*	990	739
Experteach II *by IntelligenceWare*	475	339
EXSYS *Development Software by EXSYS*	395	309
EXSYS Runtime System	600	469
Insight 1 *by Level Five Research*	95	75
Insight 2+ *by Level Five Research*	485	379
Intelligence/Compiler *by IntelligenceWare*	990	739
Logic-Line Series *All varieties by Thunderstone*	CALL	CALL

ai - lisp language

GCLISP *Golden Common LISP by Gold Hill*	495	CALL
GCLISP 286 Developer *by Gold Hill*	1190	CALL
Microsoft LISP *Common LISP*	250	149
QNIAL *Combines LISP & APL by NIAL Systems*	CALL	CALL
TransLISP PLUS *from Solution Systems*	195	125

ai - prolog language

APT *Active Prolog Tutor from Solution Systems*	65	45
Arity Combination Package	1095	979
Expert System Development Pkg	295	229
File Interchange Toolkit	50	44
PROLOG Compiler & Interpreter	650	569
Screen Design Toolkit	50	44
SQL Development Package	295	229
Arity PROLOG Interpreter	295	229
Arity Standard Prolog	95	77
LPA microPROLOG *All Varieties*	CALL	CALL
MPROLOG Language Primer *LOGICWARE*	50	45
MPROLOG P500 *by LOGICWARE*	495	395
MPROLOG P550 *by LOGICWARE*	220	175
Prolog-86 Plus *from Solution Systems*	250	159
Turbo PROLOG *by Borland Intl*	100	63
Turbo PROLOG Toolbox *by Borland Intl*	100	64

ai - smalltalk language

Combo Offer *by Digitalk* ... *Offer ends 8/31*	200	185
Smalltalk/V	99	84
EGA/VGA Color Option	50	45
Goodies Diskette	50	45
Smalltalk/Comm	50	42

ai - texas instruments

Arborist *Decision Tree Software* ... *New*	595	519
PC Scheme *Lisp* ... *New Version*	95	84
Personal Consultant Easy	495	435
Personal Consultant Images	495	435
Personal Consultant Online	995	869
Personal Consultant Plus ... *New Version*	2950	2589
Personal Consultant Runtime	95	84

ada language

AdaVantage *by Meridian Software Systems* ... *New*	795	739
Janus/ADA C Pak *by R&R Software*	95	84
Janus/ADA D Pak *by R&R Software*	1250	1059
Janus/ADA ED Pak *by R&R Software*	395	349

apl language

APL*PLUS/PC *by STSC*	595	424
APL*PLUS/PC Spreadsheet Mgr *by STSC*	195	139
APL*PLUS/PC Tools Vol 1 *by STSC*	295	199
APL*PLUS/PC Tools Vol 2 *by STSC*	85	58
ATLAS*GRAPHICS *by STSC*	450	329
Financial/Statistical Library *by STSC*	275	189
Pocket APL *by STSC*	95	69
STATGRAPHICS *by STSC*	795	579

assembly language

386 ASM/LINK *Cross Asm by Phar Lap*	495	389
8088 Assembler *w/Z-80 Translator by 2500 AD*	100	89
ASMLIB *Function Library by BC Assoc*	149	125
asmTREE *B-Tree Dev System by BC Assoc*	395	329
Cross Assemblers *Various by 2500 AD*	CALL	CALL
Microsoft Macro Assembler	150	93
Norton Utilities *by Peter Norton*	100	CALL
Norton Utilities (Advanced) ... *New*	150	99
Turbo EDITASM *by Speedware*	99	84
Uniware Cross Assemblers *Various by SDS*	CALL	CALL
Visible Computer: 8088 *by Software Masters*	80	64
Visible Computer: 80286 ... *New*	100	89

basic language

87 Software Pak *by Hauppauge*	180	149
EXIM Services Toolkit *by EXIM*	50	45
Finally *by Komputerwerks*	99	85
Inside Track *from Micro Help*	65	49
MACH 2 *by Micro Help*	69	55
MicroHelp Utilities *by MicroHelp*	59	49
Microsoft QuickBASIC ... *$20 Rebate Offer*	99	63
Peeks 'n Pokes *from MicroHelp*	45	35
QBase *Relational Database by Crescent* ... *New*	89	79
Quick-Tools *by BC Associates* ... *New*	130	109
QuickPak *by Crescent Software*	69	59
Scientific Subroutine Library *by Peerless*	125	99
Screen Sculptor *by Software Bottling*	125	91
Stay-Res *by MicroHelp*	95	73
True Basic *w/Run-time* ... *Special Price*	200	99
True Basic	100	79
Run-time Module	100	79
Various Utilities	50	41
Turbo BASIC Compiler *by Borland Intl*	100	64

blaise products

ASYNCH MANAGER *Specify C or Pascal*	175	135
C TOOLS PLUS	175	135
EXEC *Program Chainer*	95	75
LIGHT TOOLS *for Datalight C* ... *Special Price*	100	65
PASCAL TOOLS	125	99
PASCAL TOOLS 2	100	79
PASCAL TOOLS & TOOLS 2	175	135
RUNOFF *Text Formatter*	50	45
TURBO ASYNCH PLUS	100	79
TURBO C TOOLS ... *New*	129	CALL
TURBO POWER TOOLS PLUS	100	79
VIEW MANAGER *Specify C or Pascal*	275	199

borland products

EUREKA *Equation Solver*	100	64
Reflex & Reflex Workshop	200	128
Reflex *Data Base System*	150	89
Reflex Workshop	70	45
Sidekick & Traveling Sidekick	125	85
Sidekick	85	57
Traveling Sidekick	70	45
Superkey	100	64
Turbo BASIC Compiler	100	64
Turbo C Compiler *(Call for support products)*	100	64
Turbo Database Toolbox	70	41
Turbo Editor Toolbox	70	41
Turbo Gameworks Toolbox	70	41
Turbo Graphix Toolbox	70	41
Turbo Jumbo Pack	300	219
Turbo Lightning	100	64
Turbo PASCAL Numerical Methods Toolbox	100	64
Turbo PASCAL and Tutor	125	85
Turbo PASCAL	100	64
Turbo Tutor	40	24
Turbo PROLOG Compiler	100	63
Turbo PROLOG Toolbox	100	64
Word Wizard	70	47
Word Wizard and Turbo Lightning	150	94

c++

C++ *by Guidelines w/version 1.1 kernel*	195	172
PforCe++ *Function Library by Phoenix*	395	215

c compilers

C86PLUS *by Computer Innovations*	497	359
Datalight C Compiler *by Datalight*	60	43
Datalight Developer Kit	99	74
Datalight Optimum-C	139	99
DeSmet C *w/Debugger & Large case* ... *New Version*	209	184
DeSmet C *w/Debugger only* ... *New Version*	159	138
Eco-C *Complete System by Ecosoft*	140	119
Lattice C Compiler *vers. 3.2 from Lattice*	500	265
MWC Let's C w/csd ... *Special Price until 8/31*	125	54
Microsoft C Compiler w/CodeView	450	269
Microsoft QuickC Compiler ... *New*	99	63
Turbo C Compiler *by Borland*	100	64
Uniware 68000/10/20 Cross Compiler *by SDS*	CALL	CALL

c interpreters

C-terp *by Gimpel, Specify compiler*	300	219
C Trainer *with Book by Catalytix*	122	87
Instant C *by Rational Systems*	500	369
Introducing C *by Computer Innovations*	125	99
Run/C *by Age of Reason*	120	79
Run/C Professional *by Age of Reason*	250	155

c utilities

c-tree & r-tree Combo *by FairCom*	650	518
c-tree *ISAM File Manager*	395	314
r-tree *Report Generator*	295	238
C Windows *by Syscom*	100	85
C Wings *by Syscom*	50	43
Data Windows *by Magus Inc* ... *New*	CALL	CALL
dBx *dBASE to C Translator by Desktop AI*	350	299
Flash-up Windows *by Software Bottling*	90	78
GraphiC *Color version by Sci Endeavors*	350	282
GRAFLIB *by Sutrasoft*	175	159
HALO *Graphics by Media Cybernetics*	300	205
HALO Development Pkg *for Microsoft*	595	389
The HAMMER *by OES Systems*	195	129
PANEL *Forms Management by Roundhill*	295	CALL
PANEL Plus *by Roundhill*	495	CALL
PC Lint *by Gimpel Software*	139	99
PLOTHP *by Sutrasoft*	175	159
Professional C Windows *by Washburn*	89	79
Scientific Subroutine Library *by Peerless*	175	128
Vitamin C *by Creative Programming*	225	158
VC Screen *Forms Designer*	100	79
Zview *by Data Mgmt Consultants*	245	139

cobol language

COBOLspII *by Flexus*	395	329
FPLIB *for Realia COBOL by BC Associates*	149	129
Micro Focus COBOL *See Micro Focus Section*		
Microsoft COBOL *See Microsoft Section*		
Realia COBOL with RealMENU	1145	899
Realia COBOL	995	783
RealCICS	995	783
RM/COBOL *by Ryan-McFarland*	950	CALL
RM/COBOL 85 *by Ryan-McFarland*	1250	CALL
SCREENIO *by Norcom* ... *New*	400	379
screenplay *for COBOL by Flexus*	175	129

css products

Combo Package *by Custom Software Systems* ... *New*	199	175
PC/SPELL *Spelling Checker* ... *New*	49	45
PC/TOOLS *UNIX-like Utilities* ... *New*	49	45
PC/VI *vi Editor*	149	99

debuggers & profilers

386 DEBUG *Cross Debugger by Phar Lap*	195	129
Advanced Trace-86 *by Morgan Computing*	175	115
Codesifter *Profiler by David Smith*	119	85
Codesmith-86 *by Visual Age*	145	98
DSD87 *by Soft Advances*	125	79
MiniProbe *by Atron*	395	369
Periscope I *with Board by Periscope*	345	289
Periscope II *with NMI Breakout Switch*	175	139
Periscope II-X *Software only*	145	105
Periscope III *8 MHz version*	995	824
Periscope III *10 MHz version*	1095	899
The PROFILER *with Source Code by DWB*	125	89
TURBOsmith *Source debugger for Turbo Pascal*	69	59
The WATCHER *Profiler by Stony Brook*	60	51

disk utilities

Back-It *by Gazelle Systems* ... *New*	100	89
Disk Optimizer *by Softlogic Systems*	60	55
FASTBACK *by 5th Generation Systems*	179	129
XenoCopy-PC *by XenoSoft* ... *New*	80	69

CIRCLE 129 ON READER SERVICE CARD

dos utilities

Command Plus *by ESP Software*	80	69
FANSI-CONSOLE *by Hersey Micro*	75	62
Norton Commander *by Peter Norton*	75	CALL
OPAL *Shell Language by Software Factory* ... *New*	99	89
Q-DOS II *by Gazelle Systems* ... *New*	70	59
Taskview *by Sunny Hill Software*	80	55

essential products

C Utility Library	185	119
Essential Comm Library *with Debugger*	250	189
Essential Comm Library *Software Only*	185	125
Breakout Debugger Only *Any language*	125	89
Essential Graphics	250	183

forth language

CFORTH *Native Code Compiler by LMI*	300	229
Forth/83 Metacompiler *Specify Target*	750	599
PC/Forth *by Laboratory Microsystems*	150	109
PC/Forth+ *by Laboratory Microsystems*	250	199
Advanced Color Graphics Support	100	74
Enhanced Graphics Support	200	148
Intel 8087 Support	100	74
Interactive Symbolic Debugger	100	74
Native Code Optimizer	100	74
Software Floating Point	100	74
UR/Forth *by LMI*	350	279
Object Module Libraries	500	395

fortran language

50 MORE: FORTRAN *by Peerless Engr*	125	95
ACS Time Series *by Alpha Computer Service*	495	389
Essential Graphics *by Essential Software*	250	183
Forlib-Plus *by Alpha Computer Service*	70	44
FORTLIB *by Sutrasoft*	125	109
FORTRAN Addendum *by Impulse Engr*	95	85
FORTRAN Addenda *by Impulse Engr*	165	138
GRAFLIB *by Sutrasoft*	175	159
HALO *Graphics by Media Cybernetics*	300	205
I/O PRO *by MEF Environmental*	149	129
Microcompatibles Combo Package	240	215
Grafmatic	135	117
Plotmatic	135	117
Microsoft FORTRAN *w/CodeView*	450	269
No Limit *by MEF Environmental*	129	109
Numerical Analyst *by MAGUS*	295	249
PANEL *by Roundhill Computer Systems*	295	CALL
PLOTHP *by Sutrasoft*	175	159
RM/FORTRAN *by Ryan-McFarland*	595	CALL
RTC PLUS *Fortran to C by Cobalt Blue*	450	399
Scientific Subroutine Lib *by Peerless*	175	128
Statistician *by Alpha Computer Service*	295	235
Statlib.GL: *by PSI/Systems*	295	239
Statlib.TSF: *by PSI/Systems*	295	239
Strings & Things *by Alpha Computer Service*	70	45

greenleaf products

Greenleaf Comm Library	185	125
Greenleaf Data Windows Library	225	155
with Source Code	395	249
Greenleaf Functions	185	125

help utilities

HELP/Control *by MDS*	125	99
On-line Help *from Opt-Tech*	149	99
SoftScreen/HELP *by Dialectic Systems*	195	149

lattice products

Lattice C Compiler *ver 3.2 from Lattice*	500	265
with Library Source Code	900	495
C Cross Reference Generator	50	37
with Source Code	200	139
C-Food Smorgasbord *Function Library*	150	95
with Source Code	300	179
C-Sprite *Source Level Debugger*	175	119
Curses *Screen Manager*	125	85
with Source Code	250	169
dBC *Specify dBC II or dBC III*	250	169
with Source Code	500	356
dBC III Plus	750	594
with Source Code	1500	1184
LMK *Make Facility*	195	138
RPG II Combo *All three items below*	1100	875
RPG II Compiler *No Royalties*	750	625
RPG II SEU *Screen Entry Utility*	250	199
RPG II Sort/Merge	250	199
RPG II Screen Design Aid Utility	350	309
SecretDisk *File Encryption Utility*	120	88
SideTalk *Resident Communications*	120	88
SSP/PC *Scientific Subroutine Library*	350	269
Text Management Utilities	120	88
TopView Toolbasket *Function Library*	250	178
with Source Code	500	356

metagraphics products

LightWINDOW/C *for Datalight C*	95	79
FontWINDOW	95	79
FontWINDOW/PLUS	275	229
MetaWINDOW *No Royalties*	195	159
MetaWINDOW/PLUS	275	229
TurboWINDOW/C *for Turbo C*	95	79
TurboWINDOW/Pascal *for Turbo Pascal*	95	79

micro focus products

Micro Focus Level II COBOL *w/Animator*	495	395
Level II COBOL	349	279
Level II Animator	195	155
Micro Focus Level II COBOL/ET for UNIX	CALL	CALL
Micro Focus Personal COBOL	149	119
Micro Focus Professional COBOL	2000	1595
Micro Focus VS COBOL/XENIX	1495	1195
Micro Focus Support Products:		
COBOL/IQ *Ad hoc Report Writer*	495	395
COBOL/IQ *for DOS 3.X Networks*	995	795
FORMS-2	295	235
SOURCEWRITER	995	795

microport products

System V/386 Combination ... New	799	699
386 Runtime System ... New	299	259
386 Software Development System ... New	399	349
Text Preparation System ... New	199	169
386 Unlimited License Kit ... New	299	259
DOSMerge386 Run DOS and UNIX together ... New	CALL	CALL
System V/AT Combination	549	465
AT Runtime System	199	169
AT Software Development System	249	209
Text Preparation System	199	169
AT Unlimited License Kit	249	209
DOSMerge286 Run DOS and UNIX together ... New	149	125

microsoft products

Microsoft BASIC Compiler for XENIX ... New	695	419
Microsoft BASIC Interpreter for XENIX	350	209
Microsoft C Compiler with CodeView	450	269
Microsoft COBOL Compiler with COBOL Tools	700	429
for XENIX	995	609
Microsoft FORTRAN Optimizing Compiler/CodeView	450	269
Microsoft FORTRAN for XENIX	695	419
Microsoft Learning DOS	50	36
Microsoft LISP Common LISP	250	149
Microsoft MACH 10 with Mouse & Windows	549	369
Microsoft MACH 10 Board only	399	279
Microsoft Macro Assembler	150	93
Microsoft Mouse Bus Version	175	114
Microsoft Mouse Serial Version	195	124
Microsoft muMath Includes muSIMP	300	179
Microsoft Pascal Compiler	300	179
for XENIX	695	419
Microsoft QuickBASIC ... $20 Rebate Offer	99	63
Microsoft QuickC ... New	99	63
Microsoft Sort	195	125
Microsoft Windows	99	63
Microsoft Windows Development Kit	500	299
Microsoft Word	450	269

modula-2 language

EXE2LNK MASM Interface by PMI ... New	49	45
Macro2 Macro preprocessor by PMI ... New	89	79
ModBase by PMI ... New	89	79
MODULA-2 Apprentice Pkg by LOGITECH	99	79
MODULA-2 Magic Pkg by LOGITECH	99	79
MODULA-2 ROM Pkg & Cross RT Debugger	299	239
MODULA-2 Window Pkg by LOGITECH	49	39
MODULA-2 Wizard's Pkg by LOGITECH	199	159
Repertoire by PMI	89	74

mouse products

LOGIMOUSE BUS with PLUS Pkg by LOGITECH	119	98
with PLUS & PC Paintbrush	149	119
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179
LOGIMOUSE C7 with PLUS Pkg, Specify Connector	119	98
with PLUS & PC Paintbrush	149	119
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179
Microsoft Mouse Bus Version	175	114
Microsoft Mouse Serial Version	195	124

other languages

ACTOR by Whitewater Group ... New	495	419
CCS MUMPS Single-User by MGlobal	60	50
CCS MUMPS Single-User Multi-Tasking	150	129
CCS MUMPS Multi-User	450	359
Marshal Pascal by Marshal Language Systems	189	155
Pascal-2 by Oregon Software ... New	395	325
Personal REXX by Mansfield Software	125	99
SNOBOL4+ by Catspaw	95	80

other products

Dan Bricklin's Demo Pgm by Software Garden	75	57
Dan Bricklin's Demo Tutorial	50	45
Fast Forward by Mark Williams ... New	70	59
Informix All Varieties by Informix	CALL	CALL
Instant Replay by Nostradamus	150	CALL
MKS Toolkit w/vi Editor by MKS ... New Version	139	114
MicroTEX Typesetting from Addison Wesley ... New	295	CALL
Net-Tools by BC Associates	149	129
OPT-Tech Sort by Opt-Tech Data Proc	149	99
PC/TOOLS by Custom Software	49	45
Screen Machine by MicroHelp	79	59

phoenix products

Pasm86 Macro Assembler version 2.0	195	108
Pdisk Hard Disk & Backup Utility	145	89
Pfantasy Pac Phoenix Combo	995	599
Pfinish Execution Profiler	395	225
Pfix86plus Symbolic Debugger	395	225
PforCe Specify C Compiler	395	209
PforCe++ Specify C Compiler and C++	395	215
Plink86plus Overlay Linker	495	279
Pmaker Make Utility	125	78
Pmate Macro Text Editor	195	108
Pre-C Lint Utility	295	154
Ptel Binary File Transfer Program	195	108

polytron products

PolyBoost The Software Accelerator	80	64
PolyDesk III	99	72
PolyDesk III Archivist	50	42
PolyDesk III Cryptographer	50	42
PolyDesk III Talk	70	52
PolyLibrarian Library Manager	99	73
PolyLibrarian II Library Manager	149	109
PolyMake UNIX-like Make Facility	149	109
PolyShell	149	109
Polytron C Beautifier	50	42
Polytron C Library I	99	72
Polytron PowerCom Communications	139	105
PolyXREF Complete Cross Ref Utility	219	169
PolyXREF One language only	129	99
PVCS Corporate Version Control System	395	309
PVCS Personal	149	109
PVMFM Polytron Virtual Memory File Mgr	199	155

program mgmt utilities

Interactive EASYFLOW by Haventree	150	125
PrintQ by Software Directions	89	84
Quilt Computing Combo Package	250	199
QMake Program Rebuild Utility	99	79
SRMS Software Revision Mgmt System	185	159
Source Print by Aldebaran Labs ... New Version	97	75
TLIB by Burton Systems Software	100	89
Tree Diagrammer by Aldebaran Labs ... New Version	77	67

raima products

dbQUERY Single-User Query Utility	195	129
Single-User with Source Code	495	389
Multi-User	495	389
Multi-User with Source Code	990	699
dbVISTA Single-User DBMS	195	129
Single-User with Source Code	495	389
Multi-User	495	389
Multi-User with Source Code	990	699

sco products

Complete XENIX System V by SCO	1295	994
Development System	595	499
Operating System Specify XT or AT	595	499
Text Processing Package	195	144
Lyrix by SCO	595	449
SCO Professional 1-2-3 Workalike for XENIX	795	595
SCO XENIX-NET £	595	495

softcraft products

Btrieve ISAM Mgr with No Royalties	245	184
Xtrieve Query Utility	245	184
Report Option for Xtrieve	145	99
Btrieve/N for Networks	595	454
Xtrieve/N	595	454
Report Option/N for Xtrieve/N	345	269

text editors

Brief & dBrief Combo from Solution Systems	275	CALL
Brief	195	CALL
dBrief Customizes Brief for dBASE III	95	CALL
Epsilon Emacs-like editor by Lugaru	195	147
KEDIT by Mansfield Software	125	98
Micro/SPF by Phaser Systems	175	139
Microsoft Word	450	269
PC/VI by Custom Software Systems	149	99
SPF/PC by Command Technology Corp	CALL	CALL
Vedit by CompuView	150	98
Vedit Plus by CompuView	185	128

turbo pascal utilities

ALICE Interpreter by Software Channels	95	66
DOS/BIOS & Mouse Tools by Quinn-Curtis	75	67
Flash-up Windows by Software Bottling	90	78
MACH 2 for Turbo Pascal by Micro Help	69	55
MetraByte D/A Tools by Quinn-Curtis	100	89
Science & Engrg Tools by Quinn-Curtis	75	67
Screen Sculptor by Software Bottling	125	91
Speed Screen by Software Bottling	35	32
System Builder by Royal American	150	129
IMPEX Query Utility	100	89
Report Builder	130	115
TDebugPLUS by TurboPower Software	60	49
Turbo EXTENDER by TurboPower Software	85	64
Turbo OPTIMIZER by TurboPower ... New	75	65
Turbo OPTIMIZER with Source Code ... New	125	108
Turbo Professional by Sunny Hill	70	45
Turbo.ASM by BC Associates ... New	100	84
TurboHALO from IMSI	129	98
TurboPower Utilities by TurboPower	95	78
TurboRef by Gracon Services	50	45
TURBOsmith Source Debugger by Visual Age	69	59
Universal Graphics Library by Quinn-Curtis ... New	150	119

wendin products

Operating System Toolbox	99	79
PCNX Operating system	99	79
PCVMS Similar to VAX/VMS	99	79
Wendin-DOS Multitasking DOS ... New	99	85
XTC Text Editor w/Pascal source	99	75

xenix/unix products

Btrieve ISAM File Mgr by SoftCraft	595	454
C-terp by Gimpel, Specify compiler	498	379
c-tree ISAM Mgr by FairCom	395	314
dBx with Library Source by Desktop AI	550	489
DOSIX Console Version by Data Basics	399	349
DOSIX User Version by Data Basics	199	179
Informix All Varieties by Informix	CALL	CALL
Micro Focus Products See Micro Focus Section		
Microport Products See Microport Section		
Microsoft Products See Microsoft Section		
PANEL Plus by Roundhill Computer Systems	495	CALL
REAL-TOOLS Binary Version by PCT	149	89
Library Source Version	399	289
Complete Source Version	499	369
RM/COBOL by Ryan-McFarland	1250	CALL
RM/FORTRAN by Ryan-McFarland	750	CALL
SCO Products See SCO Section		

LOWEST PRICES

Due to printing lead times, some of our current prices may differ from those shown here. Call for latest pricing.

FREE SHIPPING

Orders within the USA (including Alaska & Hawaii) are shipped FREE via UPS. Express shipping is available at the shipping carrier's standard rate with no rush fees or handling charges. To avoid delays when ordering by mail, please call first to determine the exact cost of express shipping.

CREDIT CARDS

VISA and MasterCard are accepted at no extra cost. Your card is charged when your order is shipped. Mail orders please include credit card expiration date and authorized signature.

CODs AND POs

CODs and Purchase Orders are accepted at no extra cost. POs with net 30-day terms are available to qualified US accounts only.

SALES TAX

Orders outside of Ohio are not charged state sales tax. Ohio customers please add 6% Ohio tax or provide proof of tax-exemption.

INTERNATIONAL ORDERS

Shipping charges for International and Canadian orders are based on the shipping carrier's standard rate. Since rates vary between carriers, please call or write for the exact cost. International orders (except Canada), please include an additional $10 for export preparation. All payments must be made with US funds drawn on a US bank. Please include your telephone number when ordering by mail. Due to government regulations, we cannot ship to all countries.

VOLUME ORDERS

Volume orders may qualify for additional discounts. Call us for special pricing.

SOUND ADVICE

Our knowledgeable technical staff can answer technical questions, assist in comparing products and send you detailed product information tailored to your needs.

30-DAY GUARANTEE

Most of our products (excluding books) come with a 30-day documentation evaluation period or a 30-day return guarantee. Please note that some manufacturers restrict us from offering guarantees on their products. Call for more information.

MAIL ORDERS

Please include your telephone number on all mail orders. Be sure to specify computer, operating system and any applicable compiler or hardware interface(s). Send mail orders to:

Programmer's Connection
Order Processing Department
136 Sunnyside Street
Hartville, OH 44632

Call or write for our FREE comprehensive price guide.

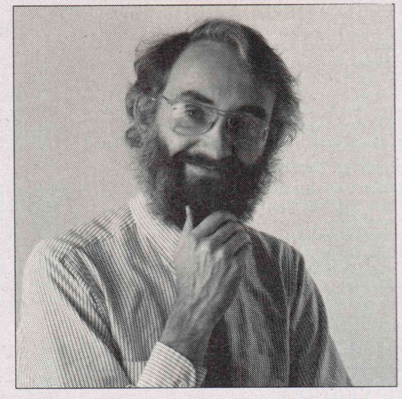
Hours: Weekdays 8:30 AM to 8:00 PM EST.

CALL TOLL FREE

USA	800-336-1166
CANADA	800-225-1166
OHIO & ALASKA (Collect)	216-877-3781
TELEX	9102406879
EASYLINK	62806530
INTERNATIONAL	216-877-3781
CUSTOMER SERVICE	216-877-1110

programmer's connection

SWAINE'S FLAMES



One database-language developer at the spring Comdex threw up his hands and squawked, "Whose idea was all this SQL stuff, anyway?" This particular developer was of the opinion that an intelligent user ought to be able to structure his own queries and that if he couldn't, he probably didn't know what he wanted to know; he opined that SQL must be something foisted on developers by Esther Dyson.

Philippe Kahn wants to know why price has ceased to be an issue worthy of discussion in the computer press. He thinks that users would be better served if more attention were paid to what they get for their dollars. In grinding the Borland Ax, Philippe makes a fine point. Why indeed would people who receive hordes of free review copies of software be insensitive to price and value?

Aussie reader Richard Walding writes to ask if I'm sure about the genesis I presented in *Fire in the Valley* of the Apple Computer bitten apple logo. "Alan Turing," he writes, "the man who laid the theoretical basis of digital computing. . .ended his life by eating an apple covered with cyanide." There seems to be some question about how the cyanide got into Turing's system, but the bitten apple was there at the scene, and it is the sort of metaphor that would appeal to Wozniak, who priced his Apple I at $666, the Biblical Number of the Beast.

Which raises the question of what logo Bill Campbell will use for his new Apple software company when Apple spins off its in-house software development as an independent company under Bill's management. A candy-coated apple?

Then there is the odd subscription inquiry:

"Regarding your reply to our letter of April 1, 1987. You wrote that we should indicate which publication we are requesting. Please note that our client is asking for:
Dr. Dobb's Journal of Computer Tools
Dr. Dobb's Journal of Modular Tools
Dr. Dobb's Journal of Computer Calisthenics"

He seems to think that in addition to *Dr. Dobb's Journal of Software Tools* (to which he is already subscribing) there are three other *Dr. Dobb's Journals*. Do you know anything about these other three *Dr. Dobb's*?

Some people, on the other hand, are all answers, like my entrepreneurial cousin Corbett. . . .

Cousin Corbett's Secrets of Software Success, Part I: The Modern Software Product Life Cycle.

The savvy software developer of today must keep pace with mercurial market conditions. The modern software product life cycle, or MSPLC, consists of the familiar phases of research, development, maintenance, and maturity, but in a streamlined form.

Research. First, you must identify an appropriate market. Size alone is not important: lots of people use computers at home, yet there is no home market; on the other hand, there are only a few hundred computer-industry journalists, yet they constitute a market vital to your success. Industry writers found laptop computers extremely useful in their work and ensured their success, and a program that simulates flying an airplane has been successful in part because of its value to reviewers in evaluating IBM ROM BIOS compatibility. So focus on products that benefit journalists. (N.B.: your time is precious and should not be squandered on nonproducts, so write no code during this phase. If the research phase indicates that your idea has market viability, announce a product and start taking orders.)

Development. Once you have begun cashing checks you legally have only three months to produce something. Many developers make the mistake of trying to do too much in this phase, resulting in missed delivery dates and buggy software. Recognize this truth: three months is not enough time to write a useful application.

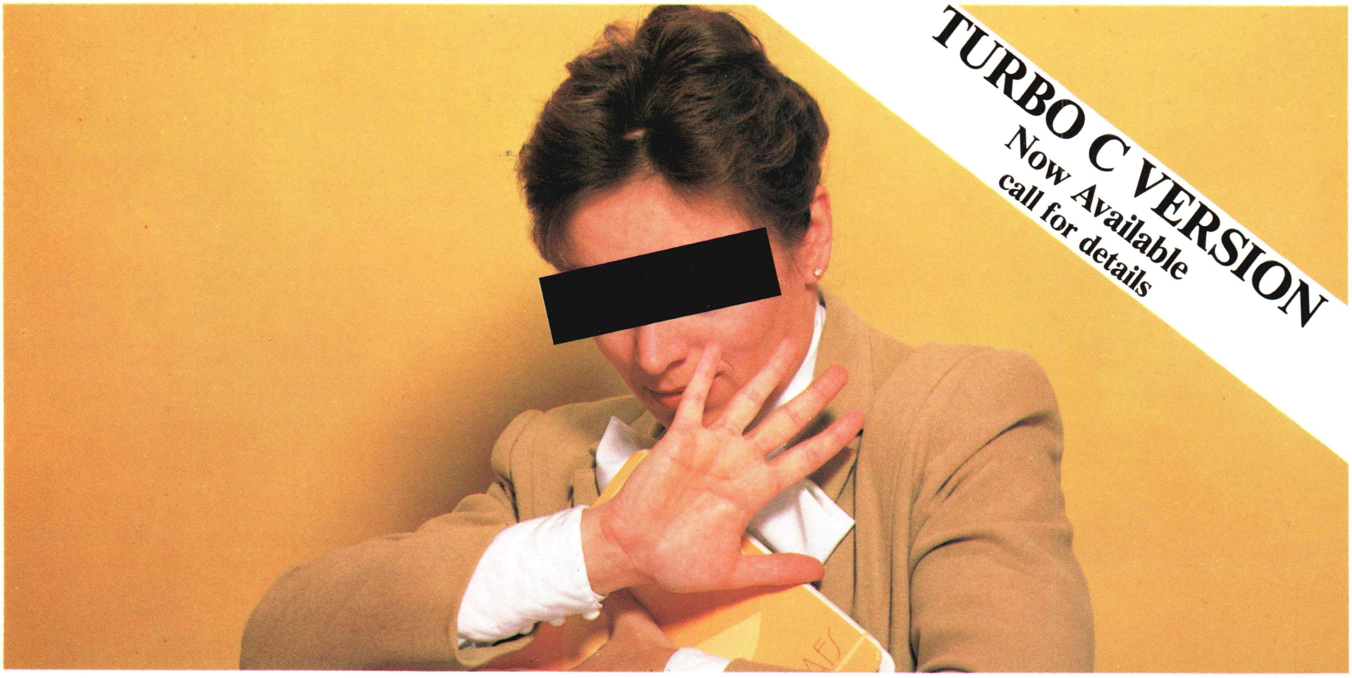
The solution? Concentrate your efforts on the user interface. Develop your version 1.0 using one of the many programs for producing demos and mockups. This will give the user something to critique, establish your "look and feel" claims, and allow you to concentrate on launching your full-scale user-funded promotional campaign.

Maintenance. In this stage, hire some coders to flesh out the product based on the useful feedback users will provide on the features they most want. Note that by postponing actual coding until this phase you have avoided the waste of creating unwanted features.

Maturity. The maintenance phase ultimately produces Release 1.1, the first working version, and marks the beginning of the mature phase of the product's life. Don't think that you can then relax, though—to keep the product viable you should produce a completely new version of the packaging at least twice a year.

Michael Swaine

Michael Swaine
editor-in-chief



You'd Be Surprised Who Uses Essential Graphics Functions

And some of our well-known customers would be just as surprised if they saw themselves in this ad. We don't splash their names all over the place as a matter of professional courtesy.

Let's face it. Just because someone else uses a product is not reason enough to buy it. The clincher is that our programs run a *documented 40% faster* than the closest competitor. To complete the picture, our code is up to *75% smaller* due to efficient coding and the granularity of functions.

However, should curiosity get the best of you, call us at 201-762-6965 and we'll drop a few highly impressive names on you.

Behind every great program is a great library.

Draw Your Own Conclusions

When you're responsible for a project that includes advanced graphics, "graphics windowing," or character font manipulation, Essential Graphics is the clear choice. We've taken the grind out of graphics programming and replaced it with speed and versatility.

No Royalties, 30 Day Guarantee

We believe that selling you a programming tool does not make us your co-authors. So we don't charge any royalties or run time fees. If within 30 days you don't find our library satisfactory, dump the whole thing and receive a complete refund.

Functions At A Glance

Features:
- Fastest functions available
- Dots, Lines, Circles, Arcs, Pies, Bars
- Manipulate character fonts
- Move blocks, do animation
- User definable patterns
- Seed filling in a boundary
- Clipping on screen coordinates

Devices Supported:
- IBM, Epson, Oki printers
- HP Plotters, HP Laser Jet
- Microsoft, Logitech Mice

Graphics Adapters:
- IBM Color Graphics
- IBM Enhanced Graphics
- Hercules Graphics
- AT&T, Olivetti Graphics
- Tecmar Graphics Master
- Others (Call)

Compiler Compatibility:
- Microsoft, C, Fortran, Pascal
- Lattice C, Aztec C
- Computer Innovations C86, DeSmet C
- Wizard C, Mark Williams

$250.00

Do Your Homework

The library you buy can influence the rest of your programming life. We encourage you to do some checking before making a decision. When you've done your homework, you'll choose Essential. Call our support staff of experienced C programmers and find out before you buy how things will be after your check clears.

To order or for support call: 201-762-6965

For foreign orders contact:

England: Gray Matter Tel. (0364) 53499
Japan: Lifeboat Inc. of Japan Tel: 293 4711
West Germany: Omnitex Tel. 07623-61820

Essential Software, Inc.
P.O. Box 1003, Maplewood, New Jersey 07040

No matter where you take CROSSTALK® Mk.4...
You won't encounter a PC communications program with as much versatility as CROSSTALK® Mk. 4. It has everything we could imagine you needing today. More protocols — X.PC, Xmodem, Kermit, and our own CROSSTALK. More terminal emulations, including complete IBM 3101, DEC VT-100, and TeleVideo 900 series. Concurrent communications capability — up to 15 sessions, each displayed in its own expandable window, or on separate "pages." Error checking at high speeds. Prepared script files to extract information from most popular information utilities. A powerful programming language to create customized scripts. Finally, we've built-in a bit of tomorrow. CROSSTALK Mk. 4 is based on a modular architecture that means we can add new capabilities by phone, as they come along. So you're getting more than today's standard in communications software. You're getting tomorrow's as well.
CROSSTALK® COMMUNICATIONS
Digital Communications Associates, Inc.
1000 Holcomb Woods Parkway
Roswell, Georgia 30076
1-800-241-6393
CROSSTALK® Mk.4
CROSSTALK is a registered trademark of Digital Communications Associates, Inc. / DEC VT-100 is a registered trademark of Digital Equipment Corp.
IBM is a registered trademark of International Business Machines Corp. / TeleVideo is a registered trademark of TeleVideo Systems, Inc. / X.PC is a trademark of Tymshare, Inc.
CIRCLE 171 ON READER SERVICE CARD